CLSVOF AS A FAST AND MASS-CONSERVING EXTENSION OF THE LEVEL-SET METHOD FOR THE SIMULATION OF TWO-PHASE FLOW PROBLEMS

M. GRIEBEL* AND M. KLITZ†

Abstract. The modeling of two-phase flows in computational fluid dynamics is still an area of active research. One popular method is the coupling of level-set and volume-of-fluid (CLSVOF), which benefits from the advantages of both approaches and results in improved mass conservation while retaining the straightforward computation of the curvature and the surface normal. Despite its popularity, details on the involved complex computational algorithms are hard to find and if found, they are mostly fragmented and inaccurate. In this article, we present all details on the CLSVOF method, which is fast and conserves mass excellently even on coarse grids. All in all, this article can be used as a comprehensive guide for an implementation of CLSVOF into existing level-set Navier-Stokes solvers on Cartesian grids in three-dimensions.

Key words. CLSVOF, two-phase flow

1. Introduction. The simulation of two-phase fluid flow requires the consideration of two immiscible fluids and the treatment of the free surface in between. A popular method for implicitly capturing free surface motion is the level-set (LS) method introduced by Osher and Sethian [25], where a smooth scalar field ϕ is advected with the flow, and the zero level-set of this field represents the interface Γ_f. Furthermore, $\phi > 0$ in the liquid and $\phi < 0$ in the gas. In our setting surface tension effects are included via the CSF method [1].

On the whole domain Ω the one-fluid continuum formulation of the two-phase Navier-Stokes equations can be written as

$$\begin{aligned} \rho(\phi)\left(\partial_t \boldsymbol{u} + (\boldsymbol{u}\cdot\nabla\boldsymbol{u})\right) + \nabla p &= \nabla\cdot(\mu(\phi)\boldsymbol{S}) - \sigma\kappa(\phi)\delta(\phi)\nabla\phi + \rho(\phi)\boldsymbol{g} \\ \nabla\cdot\boldsymbol{u} &= 0 \\ \phi_t + \boldsymbol{u}\cdot\nabla\phi &= 0 \end{aligned} \tag{1.1}$$

with time $t \in [0,T]$, fluid velocity $\boldsymbol{u} = (u,v,w)^\mathsf{T}$, pressure p, stress tensor $\boldsymbol{S} = \nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^\mathsf{T}$, surface tension σ and a volume force $\boldsymbol{g}$. In this equation, $\kappa = \nabla\cdot\boldsymbol{n}$ is the curvature with the outward normal $\boldsymbol{n} = \nabla\phi/|\nabla\phi|$, where $|\cdot|$ denotes the Euclidean norm and ϕ is the level-set function. Furthermore, $\mu(\phi)$ is the viscosity and $\rho(\phi)$ the density, and they are both defined in dependence of ϕ as in [7]. The Dirac delta functional is denoted by $\delta(\phi)$ as used in [1]; see also [7, 10, 17] for details and for appropriate boundary conditions for the velocity, pressure and level-set function.

The LS method suffers from several drawbacks. Topological changes simulated with the LS method are often under-resolved. Then, these changes only occur due to the diffusion introduced by the LS method and not because of physical necessities. Additionally, the LS function should remain a signed distance function at all times, but the LS advection distorts the interface. Therefore, a so-called reinitialization step must be performed, where the LS function is replaced by a smoother, less distorted function which has the same zero level-set. Although there are several techniques for the reinitialization of the LS function, simple reinitialization techniques introduce

*Institute for Numerical Simulation, University of Bonn & Fraunhofer Institute for Algorithms and Scientific Computing SCAI, Sankt Augustin (griebel@ins.uni-bonn.de)

†German Aerospace Center (DLR), Simulation and Software Technology, Linder Höhe, Cologne, Germany (margrit.klitz@dlr.de).

numerical diffusion to the solution. This leads to difficulties with volume conservation [6]. Therefore, several modified reinitialization methods have been developed to improve mass conservation [27, 31, 36] such as the local volume correction method by Sussman and Fatemi [35] or the global volume correction as in [7]. Unfortunately, these methods can in turn become a source of numerical errors and introduce undesirable oscillations in the curvature [11, 15, 17].

A further way to improve on the mass conservation of the LS function is the use of hybrid methods. In [8] the LS method is coupled with a Lagrangian particle approach, where the marker particles are used to rebuild the level-set in regions which are underresolved. Another hybrid approach is the coupling of the LS with the volume-of fluid (VOF) method, which results in the coupled level-set and volume-of-fluid (CLSVOF) method described in this article. Note that for realistic simulations such as plunging breaking waves or jet atomization, the CLSVOF method shows more realistic and reasonable results than the PLS method [40, 23].

A further development of the CLSVOF method is the recent coupled level-set and moment-of-fluid (CLSMOF) method, which uses next to the LS and the VOF function also a reference centroid in order to produce a slope and an intercept for the local reconstruction of the interface. Jemison et al. [14] claim similar accuracy of the CLSMOF and the CLSVOF method for many test problems and a better preserved interface with the CLSMOF method for three-dimensional (3D) problems with deforming, stretching or disintegrating interfaces. In parts, the CLSMOF implementation is based on the CLSVOF method, which we describe extensively in this article. Therefore, our description also offers a natural access to the CLSMOF method, which, additionally, requires the consideration of the centers of mass in each computational cell.

Although the idea of coupling LS with VOF methods is not new and there are many articles devoted to this topic, e.g. [22, 23, 33, 32, 36, 34, 37, 39], the vast majority of them skips the details of the involved complex computational algorithms or focuses on single parts of them only, which makes a fast and straightforward implementation of the CLSVOF method impossible. Especially, the reinitialization of the level-set function has been rather neglected in the standard literature: This quite tricky part of the method is scarcely dealt with in [36, 34]. In contrast, Son's reinitialization strategy [32] is well described but requires the back and forth rotation of the interface cells and depends on the different possibilities of how the interface cuts the 3D numerical grid cell. The implementation of a simple reinitialization strategy proposed by Wang et al. [39] fails in three dimensions [17].

The contribution of this article is as follows: We present an implementation of the complex CLSVOF method within our two-phase Navier-Stokes solver NaSt3DGPF [9, 24], which is developed at the Institute for Numerical Simulation at the University of Bonn. This implementation is faster than the conventional level-set method and conserves mass excellently even on coarse grids. In our description of the CLSVOF method, we include all details of the implementation. Therefore, this article can be used as a comprehensive guide for an implementation into existing level-set Navier-Stokes solvers, which has not been available so far. Additionally, we present an effective new technique for the reinitialization of the level-set function within the CLSVOF method. Furthermore, we also address the details of parallelization, which is neglected in the standard literature.

In summary, this article describes the necessary additions to a flow solver which already employs a LS method but lacks the VOF function. In Section 2, we summarize the discretization of the two-phase Navier-Stokes equations. In the subsequent

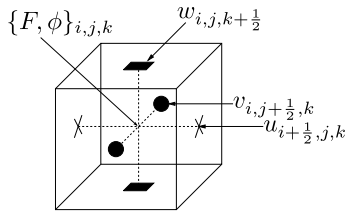


Fig. 2.1: On the staggered grid, the LS function ϕ and the VOF function F are discretized at the cell centers while the velocity components are discretized at the face centers of the grid.

Sections 3 to 7, we focus on the details of the implementation of the CLSVOF method. In Section 8, our method is evaluated by two standard test cases in two and three dimensions. We give some concluding remarks in Section 9.

2. Space-time discretization of the two-phase Navier-Stokes equations. In this section, we summarize our discretization of the level-set based two-phase Navier-Stokes equations to which the CLSVOF method is later implemented as an add-on.

We employ time stepping $t^{n+1} = t^n + \Delta t$, $n = 0, 1, \ldots, N$ where the choice of Δt ensures the stability of our discretization [7]. Specifically, we use an explicit second-order Adams-Bashforth time integration scheme. The solution process is based on the well-known projection method due to Chorin [3]: First, for each time-step n, an intermediate velocity field $\boldsymbol{u}^*$, which may not be divergence free, is advanced by the Adams-Bashforth time scheme; second, we compute a correction ∇p^{n+1} of the intermediate velocity field by the pressure Poisson equation which leads to a divergence free velocity field $\boldsymbol{u}^{n+1}$. Thus, we treat the pressure implicitly and solve the Poisson equation by a Jacobi-preconditioned conjugate gradient method.

In space, we employ a Cartesian staggered grid with grid cells

$$[x_{i-1/2}, x_{i+1/2}] \times [y_{j-1/2}, y_{j+1/2}] \times [z_{k-1/2}, z_{k+1/2}], \tag{2.1}$$

and we define the discrete computational domain Ω_h as a union of these cells. For $\{i, j, k\} \in \mathbb{Z}$ we use the notation $\delta x_i := x_{i+1/2} - x_{i-1/2}$, and δy_j as well as δz_k are defined analogously. In the following, we write $\delta x := \delta x_i$, $\delta y := \delta y_j$ and $\delta z := \delta z_k$ for the sake of shortness. Furthermore, 'ghost cells' or 'boundary cells' are needed in up to three additional strips of cells attached to Ω_h, which are necessary for the discretization of large finite difference stencils and boundary conditions.

The Navier-Stokes equations (1.1) are resolved on Ω_h by a finite difference discretization. On the staggered grid the pressure p and the LS function ϕ are discretized at the center of the cells, while the velocities u, v and w are discretized at the center of the cell faces; see Fig. 2.1. The diffusion term in (1.1) is computed by second-order central differences. A fifth-order weighted essentially non-oscillatory (WENO) scheme is used for the discretization of the convective terms in the Navier-Stokes equations. This WENO scheme is also employed for the level-set method, namely for the discretization of the transport equation's convective part in (1.1) as well as for a Hamilton-Jacobi type equation used for the purpose of reinitialization. Surface tension is evaluated using a smoothed delta function and third order interpolation. The parallelization of the code is based on conventional domain decomposition techniques using Message Passing Interface (MPI). The discretization and the solver are described in more detail in [6, 7, 17].

In what follows, we complement our LS function ϕ by the VOF function F which results in the CLSVOF method. Like ϕ, the function F tracks the interface and is

transported by

$$F_t + \boldsymbol{u} \cdot \nabla F = 0. \tag{2.2}$$

Then, as already described in the introduction, both ϕ and F contribute to a geometrical reconstruction of the interface: The smooth ϕ is used to compute the surface normal while the mass-conservative F is used to correct the mass enclosed by the zero level-set of ϕ.

Our description of the CLSVOF method is structured as follows: In Section 3, we discuss the space-time discretization of the transport equation (3.4) for which we use an operator splitting (or fractional step) method. This discussion will indicate the two main requirements for our CLSVOF method, namely the need for a geometric reconstruction of the interface (Sec. 4) to compute the VOF fluxes (Sec. 5), as well as the need for a correction of said interface to become mass conservative, which we discuss in Section 4. Furthermore, Section 6 deals with the reinitialization of the LS function. We conclude the description of the CLSVOF method with our parallelization strategy in Section 7.

3. The transport equation and its discretization. Within the level-set method the interface between the two fluids is given by the zero level-set of ϕ as $\Gamma_f(t) = \{\boldsymbol{x} : \phi(\boldsymbol{x}, t) = 0\}$ for all times $t \in [0, T]$ and $\boldsymbol{x} \in \mathbb{R}^3$, and the continuous ϕ is advected by the pure transport equation (1.1). For the CLSVOF method, we have to transport the discontinuous VOF function F. To this end, let $W \subset \Omega$ be an arbitrary small fluid volume. For $\phi > 0$ in the liquid and $\phi < 0$ in the gas, we define F as

$$F(W) := \frac{1}{|W|} \int_W H(\phi(x, y, z))\, dx\, dy\, dz \tag{3.1}$$

with the Heaviside function

$$H(\phi) := \begin{cases} 0 & \text{if } \phi < 0 \\ \frac{1}{2} & \text{if } \phi = 0 \\ 1 & \text{if } \phi > 0. \end{cases} \tag{3.2}$$

Within this integral formulation, the transport equation (2.2) becomes well-defined in a weak sense. With partial integration we obtain

$$\frac{\partial}{\partial t} \int_W H(\phi(x, y, z))\, d\boldsymbol{x} + \int_{\partial W} H(\phi(x, y, z)) \boldsymbol{u} \cdot \boldsymbol{n}\, ds = 0, \tag{3.3}$$

where $\boldsymbol{n}$ denotes the outward normal on ∂W. Thus, the change of liquid volume contained in W equals the volume flux across the boundary of W, which formally describes volume conservation and is the basis of all VOF methods [10]. After discretization, we employ (3.1) in each grid cell (2.1), so that F becomes the discontinuous liquid volume fraction on the whole domain Ω_h. Then, $F = 1$ in a cell full of liquid, $F = 0$ in a cell full of gas and $0 < F < 1$ in cells which contain the interface.

In the following, we write the transport equations (1.1) and (2.2) with $\xi = F$ or $\xi = \phi$ as

$$\frac{\partial \xi}{\partial t} + \boldsymbol{u} \cdot \nabla \xi = 0 \;\Leftrightarrow\; \frac{\partial \xi}{\partial t} + \nabla \cdot \boldsymbol{u}\xi = \xi(\nabla \cdot \boldsymbol{u}). \tag{3.4}$$

3.1. Discretization in time: Operator splitting. For the advection of the LS function ϕ and the liquid volume fraction F we use an operator splitting algorithm, i.e. we solve the transport equation (3.4) for one direction at a time. This splitting can be done in different ways and we opt for the algorithm by Son [32]. The transport equation (3.4) is discretized as

$$\frac{\xi^* - \xi^n}{\delta t} + \frac{\partial u^n \xi^n}{\partial x} = \xi^* \frac{\partial u^n}{\partial x} \tag{3.5}$$

$$\frac{\xi^{**} - \xi^*}{\delta t} + \frac{\partial v^n \xi^*}{\partial y} = \xi^* \frac{\partial v^n}{\partial y} \tag{3.6}$$

$$\frac{\xi^{n+1} - \xi^{**}}{\delta t} + \frac{\partial w^n \xi^{**}}{\partial z} = \xi^* \frac{\partial w^n}{\partial z}. \tag{3.7}$$

Here, ξ^* and ξ^{**} are functions of intermediate time in between time steps n and $n+1$. The operator splitting is of second order in time if we alternate the starting sweep direction in every time step, i.e. if we permute the sweep order by x, y, z - y, z, x - z, x, y. In the following, all our descriptions focus on the sweep order x, y, z since the use of the other two simply corresponds to a permutation of the coordinate directions.

3.2. Discretization in space. Both the LS function ϕ and the VOF function F are discretized at the center of the grid cells as depicted in Figure 2.1. Integration of the equations (3.5)–(3.7) over the computational cell (i, j, k) yields

$$\begin{aligned}
\xi^*_{i,j,k} &= \frac{\xi^n_{i,j,k} - \frac{\delta t}{\delta x}\left(G_{i+1/2,j,k} - G_{i-1/2,j,k}\right)}{1 - \frac{\delta t}{\delta x}\left(u_{i+1/2,j,k} - u_{i-1/2,j,k}\right)} \\
\xi^{**}_{i,j,k} &= \xi^*_{i,j,k}\left(1 + \frac{\delta t}{\delta y}\left(v_{i,j+1/2,k} - v_{i,j-1/2,k}\right)\right) - \frac{\delta t}{\delta y}\left(G^*_{i,j+1/2,k} - G^*_{i,j-1/2,k}\right) \\
\xi^{n+1}_{i,j,k} &= \xi^{**}_{i,j,k} + \xi^*_{i,j,k}\frac{\delta t}{\delta z}\left(w_{i,j,k+1/2} - w_{i,j,k-1/2}\right) - \frac{\delta t}{\delta z}\left(G^{**}_{i,j,k+1/2} - G^{**}_{i,j,k-1/2}\right),
\end{aligned} \tag{3.8}$$

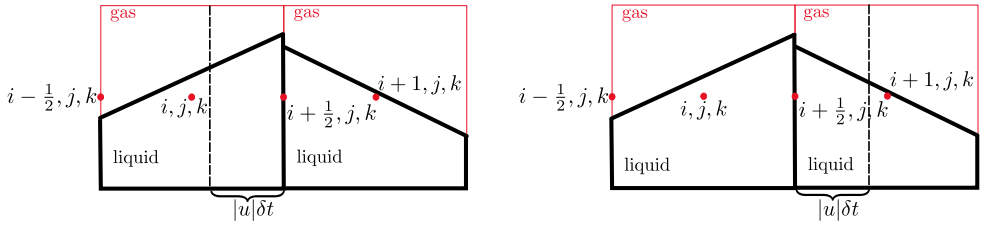
where $G_{i+1/2,j,k} := \xi_{i+1/2,j,k} u_{i+1/2,j,k}$. This is the flux of ξ across the face $(i+1/2, j, k)$ of the (i, j, k)-th computational cell. Similarly, we write $G^*_{i,j+1/2,k} := \xi^*_{i,j+1/2,k} v_{i,j+1/2,k}$ and $G^{**}_{i,j,k+1/2} := \xi^{**}_{i,j,k+1/2} w_{i,j,k+1/2}$. In the same way, $G_{i-1/2,j,k}$, $G^*_{i,j-1/2,k}$ and $G^{**}_{i,j,k-1/2}$ are defined as the fluxes across the face $(i-1/2, j, k)$, $(i, j-1/2, k)$ and $(i, j, k-1/2)$ of the (i, j, k)-th computational cell.

For the solution of these equations we have to determine $\xi_{i\pm\frac{1}{2},j,k}$, $\xi_{i,j\pm\frac{1}{2},k}$ and $\xi_{i,j,k\pm\frac{1}{2}}$. To this end, we have to distinguish between the case that ξ denotes the LS fluxes or the VOF fluxes. Since the LS function is continuous, the respective fluxes can be computed by interpolation from nearby cells. This, however, is not possible for the discontinuous VOF function, where the flux computation requires a geometric reconstruction of the interface.

In the following, we focus on $\xi_{i+\frac{1}{2},j,k}$, $\xi_{i,j+\frac{1}{2},k}$ and $\xi_{i,j,k+\frac{1}{2}}$ only since the computation of the fluxes across the downward cell faces works in an equivalent fashion.

3.2.1. Computation of the LS fluxes $\xi = \phi$. Since the LS function ϕ is smooth, $\phi_{i+1/2,j,k}$, $\phi_{i,j+1/2,k}$ and $\phi_{i,j,k+1/2}$ are obtained by extrapolation of ϕ in space and time [22, 36]:

$$\phi^n_{i+1/2,j,k} = \phi^n_{i,j,k} + \frac{\delta x}{2}\begin{cases} \left(1 - u_{i+1/2,j,k}\frac{\delta t}{\delta x}\right)\frac{\phi^n_{i+1,j,k} - \phi^n_{i-1,j,k}}{2\delta x} & \text{if } u_{i+1/2,j,k} > 0 \\ \left(1 + u_{i+1/2,j,k}\frac{\delta t}{\delta x}\right)\frac{\phi^n_{i+2,j,k} - \phi^n_{i,j,k}}{2\delta x} & \text{if } u_{i+1/2,j,k} < 0 \end{cases}$$



(a) $u_{i+1/2,j,k} > 0$ **(b)** $u_{i+1/2,j,k} < 0$

Fig. 3.1: If $u_{i+1/2,j,k} > 0$, there is flux from cell (i,j,k) to $(i+1,j,k)$. If $u_{i+1/2,j,k} < 0$, there is flux from cell $(i+1,j,k)$ to (i,j,k).

$$\phi^*_{i,j+1/2,k} = \phi^*_{i,j,k} + \frac{\delta y}{2}\begin{cases}\left(1 - v_{i,j+1/2,k}\frac{\delta t}{\delta y}\right)\frac{\phi^*_{i,j+1,k}-\phi^*_{i,j-1,k}}{2\delta y} & \text{if } v_{i,j+1/2,k} > 0\\ \left(1 + v_{i,j+1/2,k}\frac{\delta t}{\delta y}\right)\frac{\phi^*_{i,j+2,k}-\phi^*_{i,j,k}}{2\delta y} & \text{if } v_{i,j+1/2,k} < 0\end{cases}$$

$$\phi^{**}_{i,j,k+1/2} = \phi^{**}_{i,j,k} + \frac{\delta z}{2}\begin{cases}\left(1 - w_{i,j,k+1/2}\frac{\delta t}{\delta z}\right)\frac{\phi^{**}_{i,j,k+1}-\phi^{**}_{i,j,k-1}}{2\delta z} & \text{if } w_{i,j,k+1/2} > 0\\ \left(1 + w_{i,j,k+1/2}\frac{\delta t}{\delta z}\right)\frac{\phi^{**}_{i,j,k+2}-\phi^{**}_{i,j,k}}{2\delta z} & \text{if } w_{i,j,k+1/2} < 0.\end{cases}$$

Note here, however, that our solution $\phi^{n+1}_{i,j,k}$ from (3.8) is not mass-conservative. Therefore, one of the main tasks of our CLSVOF method is a correction of the interface by the volume fractions for a better conservation of mass.

3.2.2. Computation of the VOF fluxes $\xi = F$. Since F is not distributed smoothly, arithmetic interpolations from neighbor cells (as done for ϕ) are not feasible. Instead, the volume fluxes $F_{i+1/2,j,k}$, $F_{i,j+1/2,k}$ and $F_{i,j,k+1/2}$ are computed geometrically as the liquid volume fraction that is advected across a given cell face during a certain time step. This geometric computation is exemplified in Figure 3.1(a). Here, since $u_{i+1/2,j,k} > 0$, volume from cell (i,j,k) (the donor cell) is transported to cell $(i+1,j,k)$. Then, we define $F^n_{i+1/2,j,k}$ as the liquid volume fraction

$$F^n_{i+1/2,j,k} := \frac{(\delta V_F)^{n,\rightarrow}_{i+1/2,j,k}}{|u|_{i+1/2,j,k}\delta t\delta y\delta z},$$

i.e. as the liquid volume $(\delta V_F)^{n,\rightarrow}_{i+1/2,j,k}$ in relation to the total volume $|u|_{i+1/2,j,k}\delta t\delta y\delta z$, which is advected in the x-direction during the time step δt. Furthermore, the liquid volume is defined as

$$(\delta V_F)^{n,\rightarrow}_{i+1/2,j,k} := \int_{x_{i+1/2}-|u|_{i+1/2,j,k}\delta t}^{x_{i+1/2}} \int_{y_{j-1/2}}^{y_{j+1/2}} \int_{z_{k-1/2}}^{z_{k+1/2}} H\left(\phi^{R,n}_{i,j,k}(x,y,z)\right)\, dx\, dy\, dz, \tag{3.9}$$

where $\phi^{R,n}_{i,j,k}$ denotes the piecewise linear reconstruction of the interface at time $t^n = n\cdot\delta t$ and H is the Heaviside function (3.2). This, however, requires the reconstruction of the interface $\phi^{R,n}_{i,j,k}$ in the first place, which is explained in Section 4.

In the equation above and in the following, the superscript arrows $\leftrightarrows$ of (δV_F) indicate the donor cell. If the velocity is positive, there is flux from the donor cell (i,j,k) to $(i+1,j,k)$, which we depict by the arrow from left to right, because of the flux direction $(i,j,k) \rightarrow (i+1,j,k)$. On the other hand, if the velocity is negative we

have flux from the donor cell $(i+1,j,k)$ to (i,j,k), which we indicate by the arrow from right to left, because of the flux direction $(i,j,k) \leftarrow (i+1,j,k)$. Both situations are depicted in Figure 3.1.

For the sake of shortness we write $u = u_{i+1/2,j,k}$, $v = v_{i,j+1/2,k}$, $w = w_{i,j,k+1/2}$. For the solution of (3.5)–(3.7), we compute the required volume fractions as

$$\begin{aligned} F^n_{i+1/2,j,k} &= \frac{1}{|u|\delta t\delta y\delta z} \begin{cases} (\delta V_F)^{n,\rightarrow}_{i+1/2,j,k} & \text{if } u > 0 \\ (\delta V_F)^{n,\leftarrow}_{i+1/2,j,k} & \text{if } u < 0 \end{cases} \\ F^*_{i,j+1/2,k} &= \frac{1}{|v|\,\delta t\delta x\delta z} \begin{cases} (\delta V_F)^{*,\rightarrow}_{i,j+1/2,k} & \text{if } v > 0 \\ (\delta V_F)^{*,\leftarrow}_{i,j+1/2,k} & \text{if } v < 0 \end{cases} \\ F^{**}_{i,j,k+1/2} &= \frac{1}{|w|\,\delta t\delta x\delta y} \begin{cases} (\delta V_F)^{**,\rightarrow}_{i,j,k+1/2} & \text{if } w > 0 \\ (\delta V_F)^{**,\leftarrow}_{i,j,k+1/2} & \text{if } w < 0 \end{cases} . \end{aligned} \tag{3.10}$$

The computation of the advected liquid volumes such as $(\delta V_F)^{\leftrightarrows}_{i+1/2,j,k}$ with the help of the reconstructed interface is explained in Section 5.

3.3. Summary and further tasks. We have seen that the basic idea of the CLSVOF method is the construction of a new interface ϕ^R, which will be done with the help of both the VOF function F and the LS function ϕ: We want to find a piecewise linear reconstruction of the interface which is as close as possible to ϕ. For the sake of mass conservation, we then shift the interface in such a way that the area beneath the linear reconstruction equals the given volume fraction. This interface construction is performed in each operator splitting step and is then employed for the evaluation of the advected liquid volumes on the one hand and, finally, for the reinitialization of the LS function on the other hand.

For the reconstruction and advection of the interface we use the following procedure by Son [32] exemplified for the sweep direction x-y-z:

1. Use F^n and ϕ^n to reconstruct the interface in (i,j,k) for the computation of the volume flux $(\delta V_F)^n_{i\pm\frac{1}{2},j,k}$. Solve equation (3.5) with $\xi^n = F^n$ and $\xi^n = \phi^n$ for F^* and ϕ^*.
2. Use F^* and ϕ^* to reconstruct the interface in (i,j,k) for the computation of the volume flux $(\delta V_F)^*_{i,j\pm\frac{1}{2},k}$. Solve equation (3.6) with $\xi^* = F^*$ and $\xi^* = \phi^*$ for F^{**} and ϕ^{**}.
3. Use F^{**} and ϕ^{**} to reconstruct the interface in (i,j,k) for the computation of the volume flux $(\delta V_F)^{**}_{i,j,k\pm\frac{1}{2}}$. Solve equation (3.7) with $\xi^{**} = F^{**}$ and $\xi^{**} = \phi^{**}$ for F^{n+1} and ϕ^{n+1}.
4. Truncate the volume fractions by equation (6.1).
5. Reconstruct the interface from F^{n+1} and ϕ^{n+1} to reinitialize ϕ^{n+1}, i.e. the LS function is set to be the exact signed normal distance to the reconstructed interface.

All in all, this procedure requires three basic steps, which will be considered in more detail in the following sections:

- Reconstruction of an interface from a given VOF and LS function (Section 4).
- Computation of the advected liquid volume fractions $F_{i\pm\frac{1}{2},j,k}$, $F_{i,j\pm\frac{1}{2},k}$ and $F_{i,j,k\pm\frac{1}{2}}$ with the reconstructed interface (Section 5).
- Reinitialization of the LS function with the help of the reconstructed interface and truncation of the volume fractions (Section 6).

4. Reconstruction of the interface. The main parts of our interface reconstruction algorithm are as follows:

- Check in which computational cells the interface needs to be reconstructed.
- In each of these computational cells, construct a linear function ϕ^R_{ijk} which is as close as possible to the real zero LS function ϕ_{ijk}.
- Shift the interface so that the area beneath the plane equals the volume fraction F_{ijk} in that cell.

Since the reconstruction of the interface is of linear type, we first introduce the definition of a plane in three dimensions. Let $(x, y, z) \in \mathbb{R}^3$ and $a, b, c, d \in \mathbb{R}$. A plane is defined by

$$ax + by + cz + d = 0 \tag{4.1}$$

with its normal (a, b, c) and the plane's Euclidean distance d from the origin. We can *normalize* this equation by dividing with $\sqrt{1/(a^2 + b^2 + c^2)}$. Then,

$$\boldsymbol{n} := \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} = \begin{pmatrix} a/\sqrt{(a^2 + b^2 + c^2)} \\ b/\sqrt{(a^2 + b^2 + c^2)} \\ c/\sqrt{(a^2 + b^2 + c^2)} \end{pmatrix}$$

is the unit normal of the plane.

4.1. Where to reconstruct the interface. In order to check in which cells we want to reconstruct the interface, we apply a procedure similar to [22, 36, 34]:

- For all computational cells (i, j, k): Check if $0 < F_{i,j,k} < 1$.
- If additionally, $\phi_{i,j,k} \cdot \phi_{i',j',k'} < 0$ or $\phi_{i,j,k} = 0$ for some $|i-i'| \leq 1$, $|j-j'| \leq 1$ and $|k-k'| \leq 1$ with $(i-i')(j-j')(k-k') \leq 1$, reconstruct the linear interface $\phi^R_{i,j,k}$ in cell (i, j, k).

Thereby, we look at cells where the level-set function changes its sign and specifically reconstruct the interface only in those cells which contain the interface; see [17] for details.

4.2. How to reconstruct the interface. For the geometric reconstruction of the interface, we opt for 'piecewise linear interface calculation' (PLIC) [12], where the interface is approximated by straight planes perpendicular to the surface normal vector of the interface in each cell. To this end, we compute a piecewise linear reconstructed LS function

$$\phi^R_{ijk}(x, y, z) := a_{i,j,k}(x - x_i) + b_{i,j,k}(y - y_j) + c_{i,j,k}(z - z_k) + d_{i,j,k}. \tag{4.2}$$

Here, $a_{i,j,k}$, $b_{i,j,k}$, $c_{i,j,k}$ are the coordinates of the vector which is normal to the plane given by

$$a_{i,j,k}(x - x_i) + b_{i,j,k}(y - y_j) + c_{i,j,k}(z - z_k) + d_{i,j,k} = 0, \tag{4.3}$$

and the intercept $d_{i,j,k}$ is the distance of the interface to the cell center (x_i, y_j, z_k).

Now, the coefficients $a_{i,j,k}$, $b_{i,j,k}$, $c_{i,j,k}$ and $d_{i,j,k}$ are determined so that the reconstruction ϕ^R_{ijk} is as close as possible to the real zero LS function ϕ [22, 36, 23]. Thus, they minimize the error functional

$$E_{i,j,k} := \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} \int_{z_{k-\frac{1}{2}}}^{z_{k+\frac{1}{2}}} H'(\phi_{i,j,k}) \\ (\phi_{i,j,k} - a_{i,j,k}(x - x_i) - b_{i,j,k}(y - y_j) - c_{i,j,k}(z - z_k) - d_{i,j,k})^2 \, dx \, dy \, dz.$$

with H' the derivative of the Heaviside function.
In order to determine a, b, c and d we minimize the discretized error

$$E^{\delta}_{i,j,k} = \sum_{i'=i-1}^{i+1} \sum_{j'=j-1}^{j+1} \sum_{k'=k-1}^{k+1} [w_{i'-i,j'-j,k'-k} H'_{\varepsilon}(\phi_{i',j',k'}) \\ (\phi_{i',j',k'} - a_{i,j,k}(x_{i'} - x_i) - b_{i,j,k}(y_{j'} - y_j) - c_{i,j,k}(z_{k'} - z_k) - d_{i,j,k})^2] ,$$

with the weights $w_{i'-i,j'-j,k'-k} = 52$ for $i = i'$, $j = j'$, $k = k'$, and $w_{i'-i,j'-j,k'-k} = 1$ otherwise. Thereby, the cell center has twice more influence than its 26 surrounding neighbors together. H'_{ε} corresponds to the smoothed delta function with thickness ε as defined in [7]. Minimization of the above error functional leads to the conditions

$$\frac{\partial E^{\delta}_{i,j,k}}{\partial a_{i,j,k}} = \frac{\partial E^{\delta}_{i,j,k}}{\partial b_{i,j,k}} = \frac{\partial E^{\delta}_{i,j,k}}{\partial c_{i,j,k}} = 0.$$

With a notation similar to [23], we write

$$\sum := \sum_{i'=i-1}^{i+1} \sum_{j'=j-1}^{j+1} \sum_{k'=k-1}^{k+1}, \qquad wh := w_{i'-i,j'-j,k'-k} H'_{\varepsilon}(\phi_{i',j',k'})$$
$$\phi := \phi_{i',j',k'}, \qquad X := (x_{i'} - x_i), \qquad Y := (y_{j'} - y_j), \qquad Z := (z_{k'} - z_k)$$

and the resulting system of equations

$$\begin{bmatrix} \sum whX^2 & \sum whXY & \sum whXZ & \sum whX \\ \sum whXY & \sum whY^2 & \sum whYZ & \sum whY \\ \sum whXZ & \sum whYZ & \sum whZ^2 & \sum whZ \\ \sum whX & \sum whY & \sum whZ & \sum wh \end{bmatrix} \begin{bmatrix} a_{i,j,k} \\ b_{i,j,k} \\ c_{i,j,k} \\ d_{i,j,k} \end{bmatrix} = \begin{bmatrix} \sum wh\phi X \\ \sum wh\phi Y \\ \sum wh\phi Z \\ \sum wh\phi \end{bmatrix}. \qquad (4.4)$$

In order to reduce the solution process to the application of two triangular systems of equations, we employ a simple Cholesky decomposition [28]. If the matrix on the left hand side of (4.4) is not positive definite (which can be the result of roundoff errors), we compute a, b, c by the central differences

$$a_{i,j,k} = \frac{\phi_{i+1,j,k} - \phi_{i-1,j,k}}{\frac{1}{2}(\delta x_{i-1} + \delta x_{i+1}) + \delta x_i} \qquad b_{i,j,k} = \frac{\phi_{i,j+1,k} - \phi_{i,j-1,k}}{\frac{1}{2}(\delta y_{j-1} + \delta y_{j+1}) + \delta y_j}$$
$$c_{i,j,k} = \frac{\phi_{i,j,k+1} - \phi_{i,j,k-1}}{\frac{1}{2}(\delta z_{k-1} + \delta z_{k+1}) + \delta z_k} \qquad \text{and} \qquad d_{i,j,k} = \phi_{i,j,k}.$$

At the boundary, one-sided differences are employed. Note that this simpler computation of the normal is less accurate in numerical tests than the solution of (4.4) [36].

After we have determined $a_{i,j,k}$, $b_{i,j,k}$, $c_{i,j,k}$ and $d_{i,j,k}$, we normalize the interface equation (4.3) by multiplication with $1/\sqrt{a^2_{i,j,k} + b^2_{i,j,k} + c^2_{i,j,k}}$. In the following, we write n_x, n_y, n_z, d instead of $a_{i,j,k}$, $b_{i,j,k}$, $c_{i,j,k}$, $d_{i,j,k}$, if the components are already normalized and if there is no confusion concerning the indices.

For the above reconstruction we have used the LS function ϕ only. In the next step, we use the VOF function F for the correction of the intercept d. For the sake of mass conservation, we correct d in such a way that the plane represented by (4.3) cuts out the same volume in the cell (i, j, k) as specified by $F_{i,j,k}$.

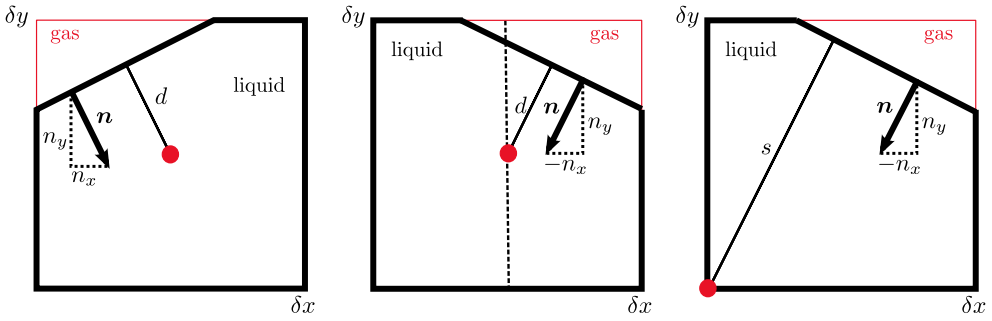


Fig. 4.1: Computing the reflected line and choosing the origin of the computational cell in 2D. The gas volume is surrounded by red solid lines and the liquid volume by thick black lines. In the first step, we reflect the line (i.e. the interface between gas and liquid) in such a way that all components of the normal vector become negative. In the above case, only $n_x > 0$ and we compute the reflection across the dashed vertical line. In the second step, we shift the origin from the cell center (distance d) to the lower left hand cell edge (distance s), which then lies in the liquid region and as far from the interface as possible.

4.3. Shifting the interface for mass conservation. Since we want to shift the interface in such a way that the area beneath the plane equals the given volume fraction, we have to formulate an algorithm which determines the distance from the plane to the origin from given $F = F_{i,j,k}$ and $\boldsymbol{n} = (n_x, n_y, n_z)^\mathsf{T}$ in each grid cell. Here, the main difficulty lies in the vast number of interface configurations which are possible in both two and three dimensions. Therefore, many authors [23, 32, 33, 36] first reduce the number of the considered interface configurations as follows [33, p. 530]:

> *The interface location is determined effectively by introducing s which denotes the distance from the interface to the corner of an interface cell [20]. The corner is chosen to be inside the liquid region and the farthest from the interface.*

Here, s denotes the distance from the interface to the corner of an interface cell. Note that the source given by [20] is 'M. Sussman, personal communication, 2001'. Further details can be found in Ménard's thesis [22] and his subsequent article [23]. Since the reduction of the number of interface configurations is a crucial part of the CLSVOF algorithm, the following subsection is merely devoted to a rigorous explanation of the above quotation.

4.3.1. Reduction of the number of interface configurations. In order to reduce the number of possible interface configurations, we make the following two transformations, which are illustrated in Figure 4.1 for the 2D case:

1. We reflect the plane in such a way that all components of its unit normal $\boldsymbol{n} = (n_x, n_y, n_z)^\mathsf{T}$ become negative.
2. We shift the plane's origin in our computational cell from the cell center to the lower left hand corner of the cell. This corner will lie in the liquid region and will be farthest away from the interface.

Our original plane as written in equation (4.3) is given by

$$n_x(x - x_i) + n_y(y - y_j) + n_z(z - z_k) + d = 0, \tag{4.5}$$

where d is the distance to the cell center (x_i, y_j, z_k). In the following, we simply substitute $\bar{x} := x - x_i$, $\bar{y} := y - y_j$ and $\bar{z} := z - z_k$ and obtain

$$n_x\bar{x} + n_y\bar{y} + n_z\bar{z} + d = 0 \tag{4.6}$$

In the first step, we reflect the plane in our computational cell in such a way that all components of the unit normal become negative (see Fig. 4.1). This requires the construction of a reflection $\mathrm{Ref} := (\mathrm{Ref}^x, \mathrm{Ref}^y, \mathrm{Ref}^z)$ with the following properties:

$$\mathrm{Ref}^x(\bar{x}) = \begin{cases} -\bar{x} & \text{if } n_x > 0 \\ \bar{x} & \text{else} \end{cases}, \mathrm{Ref}^y(\bar{y}) = \begin{cases} -\bar{y} & \text{if } n_y > 0 \\ \bar{y} & \text{else} \end{cases}, \mathrm{Ref}^z(\bar{z}) = \begin{cases} -\bar{z} & \text{if } n_z > 0 \\ \bar{z} & \text{else.} \end{cases} \tag{4.7}$$

In general, a reflection across the hyperplane which lies in the origin (so far the cell center) and is orthogonal to an arbitrary vector $\boldsymbol{e}$ is given by

$$\mathrm{Ref}_{\boldsymbol{e}}(\bar{\boldsymbol{x}}) := \bar{\boldsymbol{x}} - 2\frac{\bar{\boldsymbol{x}} \cdot \boldsymbol{e}}{\boldsymbol{e} \cdot \boldsymbol{e}}\boldsymbol{e}.$$

We choose the hyperplane in such a way that

$$\boldsymbol{e}^1 = \begin{cases} (1,0,0)^\mathsf{T} & \text{if } n_x > 0 \\ (0,0,0)^\mathsf{T} & \text{else,} \end{cases} \quad \boldsymbol{e}^2 = \begin{cases} (0,1,0)^\mathsf{T} & \text{if } n_y > 0 \\ (0,0,0)^\mathsf{T} & \text{else,} \end{cases} \quad \boldsymbol{e}^3 = \begin{cases} (0,0,1)^\mathsf{T} & \text{if } n_z > 0 \\ (0,0,0)^\mathsf{T} & \text{else.} \end{cases}$$

The successive computation of $\mathrm{Ref}_{\boldsymbol{e}^3}(\mathrm{Ref}_{\boldsymbol{e}^2}(\mathrm{Ref}_{\boldsymbol{e}^1}(\bar{\boldsymbol{x}})))$ yields the desired result (4.7). Then, the reflected plane can be written as

$$n_x\mathrm{Ref}^x(\bar{x}) + n_y\mathrm{Ref}^y(\bar{y}) + n_z\mathrm{Ref}^y(\bar{y}) + d = 0, \tag{4.8}$$

which is equal to

$$-|n_x|\,\bar{x} - |n_y|\,\bar{y} - |n_z|\,\bar{z} + d = 0, \tag{4.9}$$

since e.g. for the x-component $n_x\mathrm{Ref}^x(\bar{x}) = -|n_x|\,\bar{x}$ for all n_x.

In the second step, we place the origin in the lower left hand corner of the computational cell (see Fig. 4.1), which ensures that the plane's origin lies in the liquid region. For instance, this step would be crucial if the liquid and gas phase in Fig. 4.1 were switched. After the reflection, the origin of d would not lie in the liquid region but the origin of s will. To this end, we introduce the new coordinates $\bar{\bar{x}} := \bar{x} + 0.5\delta x$, $\bar{\bar{y}} := \bar{y} + 0.5\delta y$ and $\bar{\bar{z}} := \bar{z} + 0.5\delta z$. This shift is illustrated in Figure 4.2 for the two-dimensional case.

Substituted in (4.9), we have

$$\begin{aligned} & -|n_x|\,(\bar{\bar{x}} - 0.5\delta x) - |n_y|\,(\bar{\bar{y}} - 0.5\delta x) - |n_z|\,(\bar{\bar{z}} - 0.5\delta z) + d = 0 \\ \Leftrightarrow \quad & -|n_x|\,\bar{\bar{x}} - |n_y|\,\bar{\bar{y}} - |n_z|\,\bar{\bar{z}} \underbrace{+ |n_x|\,0.5\delta x + |n_y|\,0.5\delta x + |n_z|\,0.5\delta z + d}_{=:s} = 0, \end{aligned}$$

so that our plane in the new coordinate system is given by

$$-|n_x|\,\bar{\bar{x}} - |n_y|\,\bar{\bar{y}} - |n_z|\,\bar{\bar{z}} + s = 0, \tag{4.10}$$

and the distance s is measured from the plane to the lower left hand corner of the computational cell.

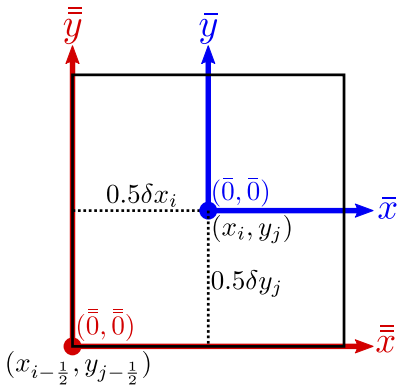


Fig. 4.2: Translation of the coordinate system exemplified in 2D: The origin is placed from the middle (x_i, y_j) to the lower left hand corner $(x_{i-\frac{1}{2}}, y_{j-\frac{1}{2}})$ of the computational cell with indices i, j, k. This corresponds to a translation where the new coordinates are expressed as $\bar{\bar{x}} := \bar{x} + 0.5\delta x_i$ and $\bar{\bar{y}} := \bar{y} + 0.5\delta y_j$.

To summarize, we first employ a reflection to ensure that all components of the plane's unit normal become negative, i.e. the unit normal points towards the lower left hand corner of the cell. Second, we shift the origin from the center of the cell to said lower left hand corner. These two steps ensure that the origin is in the liquid region and as far from the interface as possible (cf. Fig. 4.1).

All in all, we now can always consider $|n_x|, |n_y|, |n_z|$ instead of the signed components, which simplifies the computation of the liquid volumes considerably. In 2D, for example, only four configurations have to be considered for the computation of the liquid volume as demonstrated in Figure 4.3.

4.3.2. Computation of distance from given normal and volume fraction. We now describe the shift of the interface, which ensures that the liquid volume enclosed by the interface equals the given volume fraction. Due to the reduction of interface configurations, we are able to formulate an algorithm which determines the distance s of the interface from given $\boldsymbol{n}$ and F, i.e. we are able to determine how to shift the plane along its normal $\boldsymbol{n}$ to achieve a certain volume fraction F.

Before we describe the three-dimensional case, let us look the simpler two-dimensional problem as described in [32]. Let δx_0 and δy_0 denote the x- and y-intercept of the interface; see Figure 4.3. Then, in 2D the liquid volume for all possible interface configurations can be expressed by

$$\underbrace{F_{i,j,k}\delta x\delta y}_{\substack{\text{fluid volume}\\ \text{in the cell}}} = \underbrace{\frac{1}{2}\delta x_0\delta y_0}_{\substack{\text{area of}\\ \text{triangle}}} - \underbrace{\frac{1}{2}\frac{\delta y_0}{\delta x_0}\left\langle \delta x_0 - \delta x\right\rangle^2}_{\substack{\text{overlap of triangle}\\ \text{in } x\text{-direction}}} - \underbrace{\frac{1}{2}\frac{\delta x_0}{\delta y_0}\left\langle \delta y_0 - \delta y\right\rangle^2}_{\substack{\text{overlap of triangle}\\ \text{in } y\text{-direction}}} \tag{4.11}$$

which is a summation over all triangles depicted in Figure 4.3. Here, $\langle\cdot\rangle := \max(\cdot, 0)$.

Similarly, the volume of the gas phase can be written as a sum of triangles. Then, the main idea is to find a *unified* formula for the computation of the *minimum* volume, be it liquid or gas. If we compute the smaller of the liquid and gas volume, only two of the four interface configurations in Figure 4.3 are of relevance. For instance, the interface configuration on the left hand side for the computation of the liquid volume is equal to the interface configuration on the right hand side for the gas volume. In three dimensions, where there are many more possible configurations, these considerations are of great help.

Let us now describe the 3D case. In the following, we suppose that all coordinate transformations of Subsection 4.3.1 have already taken place. Thus, we always

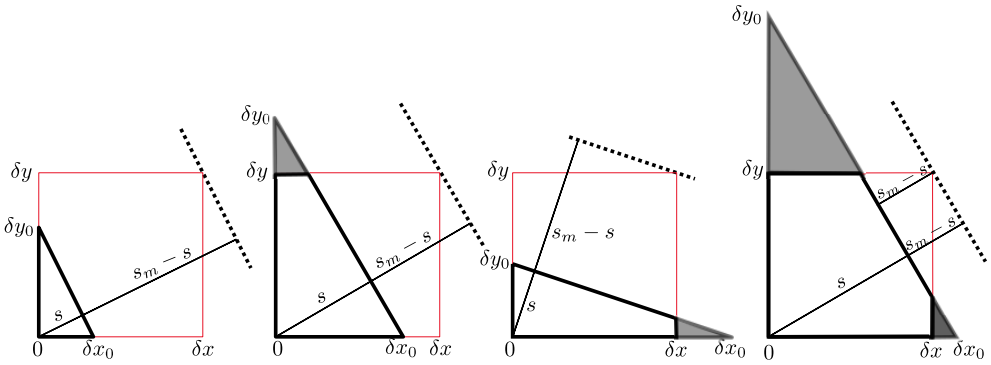


Fig. 4.3: Four cases of possible interface configurations in 2D as given in [32]. The gas volume is surrounded by red solid lines and the liquid volume by thick black lines. The line $s = |n_x|x + |n_y|y$ has the distance s to the origin, while the dashed line $s_m = \delta x_1 + \delta x_2 = |n_x|x + |n_y|y$ has the distance s_m to the origin. The volume is computed in such a way that in the last case the gas volume would be computed (with distance $s_m - s$ to the origin). δx_0 and δy_0 mark the x- and y-intercepts of the line. δx and δy are the widths of the cell. Overlapping parts of the triangles are shaded gray.

use (4.10) to describe our plane, which we write as

$$|n_x|\,x + |n_y|\,y + |n_z|\,z = s \tag{4.12}$$

for better readability.

A main requirement of the algorithm is the condition $\delta x_1 := |n_x|\delta x \geq \delta y_1 := |n_y|\delta y \geq \delta z_1 := |n_z|\delta z$ for which we do not really rotate the cell, but rename the coordinate axes if necessary. Furthermore, we define $s_m := \delta x_1 + \delta x_2 + \delta z_2$, $s_c := \min(s, s_m - s)$ and $F_c := \min(F_{i,j,k}, 1 - F_{i,j,k})$. Then, the unified formula for the computation of the liquid or vapor volume F_c in 3D is

$$6F_c\delta x_1\delta y_1\delta z_1 = s_c^3 - \langle s_c - \delta z_1\rangle^3 - \langle s_c - \delta y_1\rangle^3 - \langle s_c - \delta x_1\rangle^3 + \langle s_c - \delta y_1 - \delta z_1\rangle^3 \tag{4.13}$$

with $\langle\cdot\rangle := \max(\cdot, 0)$ [32]. For details we refer to [32] and to [17] for further explanations. This formula gives us the opportunity to compute s from given F and $\boldsymbol{n}$ on the one hand or to compute the volume fractions F from given $\boldsymbol{n}$ and s on the other hand. The latter is used in the next section for the computation of the advected liquid volume fractions.

In [32] the solution of (4.13) for s yields the following algorithm:

Input: F, n_x, n_y, n_z, δx, δy, δz. **Output:** s.

1. Set $F_c = \min(F, 1 - F)$ and rotate the interface cell so that $\delta x_1 \geq \delta y_1 \geq \delta z_1$.
2. Set $s_c = (6F_c\delta x_1\delta y_1\delta z_1)^{\frac{1}{3}}$. If$s_c < \delta z_1$ go to step 6.
3. Set $s_c = 0.5\delta z_1 + \sqrt{2F_c\delta x_1\delta y_1 - \delta z_1^2/12}$. If $s_c < \delta y_1$ go to step 6.
4. If $\delta x_1 \geq \delta y_1 + \delta z_1$, then $s_c = F_c\delta x_1 + \frac{\delta y_1 + \delta z_1}{2}$. If $s_c \geq \delta y_1 + \delta z_1$ go to step 6.
5. With the initial value s_c set by one of the above steps, solve the following equation iteratively using the Newton iteration method:

$$6F_c\delta x_1\delta y_1\delta z_1 - s_c^3 + (s_c - \delta z_1)^3 + (s_c - \delta y_1)^3 + \langle s_c - \delta x_1\rangle^3 = 0.$$

Since $\langle s_c - \delta x_1\rangle^3 = \max(s_c - \delta x_1, 0)^3$, we distinguish in our implementation between the following cases:

- $\delta x_1 \leq s_c$: Solve

$$6F_c\delta x_1\delta y_1\delta z_1 - s_c^3 + (s_c - \delta z_1)^3 + (s_c - \delta y_1)^3 + (s_c - \delta x_1)^3 = 0$$

  iteratively and go to step 6.
- $\delta x_1 > s_c$: Solve

$$6F_c\delta x_1\delta y_1\delta z_1 - s_c^3 + (s_c - \delta z_1)^3 + (s_c - \delta y_1)^3 = 0$$

  iteratively and go to step 6.

6. If $F_{i,j,k} \leq 0.5$ set $s = s_c$ else set $s = s_m - s_c$.

This algorithm concludes the geometric reconstruction of the interface. In summary, we have used the LS function ϕ to obtain a linear reconstruction which is very close to the real zero level-set of ϕ, and we have used the VOF function F for a better conservation of mass by shifting the interface. With the help of the reconstructed interface, we are now able to compute the advected liquid volume fractions in equation (3.10) since (4.13) allows us to evaluate the advected liquid volume (δV_F) geometrically as explained in the next section.

5. Computation of the advected liquid volume fractions from the reconstructed interface. In addition to the computation of the distance from normal and volume fractions, relation (4.13) also allows us to compute the volume fraction $F_{i,j,k}$ from the position of the interface in each cell. In addition, we are now able to compute the *advected* volume fractions $F_{i+1/2,j,k}$, $F_{i,j+1/2,k}$ and $F_{i,j,k+1/2}$ in (3.10).

To this end, we define the liquid volume

$$\widetilde{\delta V_F}_{i,j,k}(\boldsymbol{n}, s, \delta\tilde{x}, \delta\tilde{y}, \delta\tilde{z}) := \int_0^{\delta\tilde{x}} \int_0^{\delta\tilde{y}} \int_0^{\delta\tilde{z}} H\left(\phi^R_{i,j,k}(x,y,z)\right)\, dx\, dy\, dz, \tag{5.1}$$

which is the liquid volume in the subdomain $\delta\tilde{x} \times \delta\tilde{y} \times \delta\tilde{z}$ of the cell (i,j,k) with $0 \leq \delta\tilde{x} \leq \delta x$, $0 \leq \delta\tilde{y} \leq \delta y$ and $0 \leq \delta\tilde{z} \leq \delta z$. Again, we omit the index (i,j,k) for better readability. In contrast to $(\delta V_F)^{\rightleftarrows}$ in (3.9), the liquid volume $\widetilde{\delta V_F}$ always contains the origin of s, which is the *main requirement* for our algorithm to work. If this is the case, we can simply replace δx, δy and δz with $\delta\tilde{x}$, $\delta\tilde{y}$ and $\delta\tilde{z}$ in (4.13) and obtain the following algorithm as described in [32]:

Input: n_x, n_y, n_z, s, $\delta\tilde{x}$, $\delta\tilde{y}$, $\delta\tilde{z}$. **Output:** $\widetilde{\delta V_F}$.

1. Rename the axes so that $\delta\tilde{x}_1 := |n_x|\delta\tilde{x} \geq \delta\tilde{y}_1 := |n_y|\delta\tilde{y} \geq \delta\tilde{z}_1 := |n_z|\delta\tilde{z}$.
2. Set $\tilde{s}_m =: \delta\tilde{x}_1 + \delta\tilde{y}_1 + \delta\tilde{z}_1$ and $\tilde{s}_c := \min(s, \tilde{s}_m - s)$.
3. If $\tilde{s}_m \leq s$, the whole subregion is filled with liquid and we set $F_c = 1$. If $s < 0$, set $F_c = 0$.
4. We obtain the fluid volume fraction in the subregion by the following distinctions:
   (a) If $s_c < \delta\tilde{z}_1$, $F_c = \frac{s_c^3}{6\delta\tilde{x}_1\delta\tilde{y}_1\delta\tilde{z}_1}$.
   (b) If $\delta\tilde{z}_1 \leq s_c < \delta\tilde{y}_1$, $F_c = \frac{s_c^2 - \delta\tilde{z}_1 s_c + \delta\tilde{z}_1^2/3}{2\delta\tilde{x}_1\delta\tilde{y}_1}$, which is the solution of $6F_c\delta\tilde{x}_1\delta\tilde{y}_1\delta\tilde{z}_1 = s_c^3 - (s_c - \delta\tilde{z}_1)^3$.
   (c) If $\delta\tilde{y}_1 + \delta\tilde{z}_1 \leq s_c \leq \delta\tilde{x}_1$, $F_c = \frac{2s_c - \delta\tilde{y}_1 - \delta\tilde{z}_1}{2\delta\tilde{x}_1}$, which is the solution of $6F_c\delta\tilde{x}_1\delta\tilde{y}_1\delta\tilde{z}_1 = s_c^3 - (s_c - \delta\tilde{z}_1)^3 - (s_c - \delta\tilde{y}_1)^3 + (s_c - \delta\tilde{y}_1 - \delta\tilde{z}_1)^3$.
   (d) Otherwise, $F_c = \frac{s_c^3 - (s_c - \delta\tilde{z}_1)^3 - (s_c - \delta\tilde{y}_1)^3 - \langle s_c - \delta\tilde{x}_1\rangle^3}{6\delta\tilde{x}_1\delta\tilde{y}_1\delta\tilde{z}_1}$.
5. If $s \leq 0.5\tilde{s}_m$, set $\widetilde{\delta V_F} = F_c\delta\tilde{x}\delta\tilde{y}\delta\tilde{z}$. Otherwise, set $\widetilde{\delta V_F} = (1 - F_c)\delta\tilde{x}\delta\tilde{y}\delta\tilde{z}$.

Note that for a single-phase cell where $F = 0$ or 1, we simply set $\widetilde{\delta V_F} = F\delta\tilde{x}\delta\tilde{y}\delta\tilde{z}$ in (5.2).

This algorithm enables us to compute the liquid volume in an arbitrary rectangular part of the computational cell (i, j, k), as long as this part contains the origin of s. Note that this algorithm is also intended for the initialization of F at $t = 0$. Since the LS values are given by an analytic distance function or by an initial reinitialization step over all grid cells, we can use these distance values to compute the volume fractions. Then, in the above algorithm, $\delta\tilde{x} = \delta x$, $\delta\tilde{y} = \delta y$ and $\delta\tilde{z} = \delta z$. In the same way, this algorithm can be used for the computation of boundary values for the VOF function if a boundary cell contains a reconstructed interface.

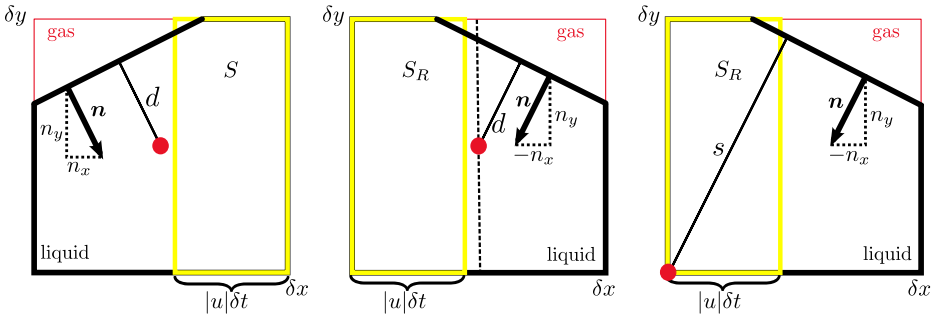
Furthermore, we are now able to compute the advected liquid volumes as follows:

$$
\begin{aligned}
(\delta V_F)^{\rightarrow}_{i+1/2,j,k} &= \left\{\begin{array}{ll} \widetilde{\delta V_F}(\boldsymbol{n}, s, |u|\,\delta t, \delta y, \delta z) & \text{if } n_x u \geq 0 \\ F\delta x\delta y\delta z - \widetilde{\delta V_F}(\boldsymbol{n}, s, \delta x - |u|\delta t, \delta y, \delta z) & \text{if } n_x u < 0 \end{array}\right\} \text{in } (i, j, k) \\
(\delta V_F)^{\leftarrow}_{i+1/2,j,k} &= \left\{\begin{array}{ll} \widetilde{\delta V_F}(\boldsymbol{n}, s, |u|\,\delta t, \delta y, \delta z) & \text{if } n_x u \geq 0 \\ F\delta x\delta y\delta z - \widetilde{\delta V_F}(\boldsymbol{n}, s, \delta x - |u|\delta t, \delta y, \delta z) & \text{if } n_x u < 0 \end{array}\right\} \text{in } (i+1, j, k) \\
(\delta V_F)^{\rightarrow}_{i,j+1/2,k} &= \left\{\begin{array}{ll} \widetilde{\delta V_F}(\boldsymbol{n}, s, \delta x, |v|\delta t, \delta z) & \text{if } n_y v \geq 0 \\ F\delta x\delta y\delta z - \widetilde{\delta V_F}(\boldsymbol{n}, s, \delta x, \delta y - |v|\delta t, \delta z) & \text{if } n_y v < 0 \end{array}\right\} \text{in } (i, j, k) \\
(\delta V_F)^{\leftarrow}_{i,j+1/2,k} &= \left\{\begin{array}{ll} \widetilde{\delta V_F}(\boldsymbol{n}, s, \delta x, |v|\delta t, \delta z) & \text{if } n_y v \geq 0 \\ F\delta x\delta y\delta z - \widetilde{\delta V_F}(\boldsymbol{n}, s, \delta x, \delta y - |v|\delta t, \delta z) & \text{if } n_y v < 0 \end{array}\right\} \text{in } (i, j+1, k) \\
(\delta V_F)^{\rightarrow}_{i,j,k+1/2} &= \left\{\begin{array}{ll} \widetilde{\delta V_F}(\boldsymbol{n}, s, \delta x, \delta y, |w|\delta t) & \text{if } n_z w \geq 0 \\ F\delta x\delta y\delta z - \widetilde{\delta V_F}(\boldsymbol{n}, s, \delta x, \delta y, \delta z - |w|\delta t) & \text{if } n_z w < 0 \end{array}\right\} \text{in } (i, j, k) \\
(\delta V_F)^{\leftarrow}_{i,j,k+1/2} &= \left\{\begin{array}{ll} \widetilde{\delta V_F}(\boldsymbol{n}, s, \delta x, \delta y, |w|\delta t) & \text{if } n_z w \geq 0 \\ F\delta x\delta y\delta z - \widetilde{\delta V_F}(\boldsymbol{n}, s, \delta x, \delta y, \delta z - |w|\delta t) & \text{if } n_z w < 0 \end{array}\right\} \text{in } (i, j, k+1).
\end{aligned}
\tag{5.2}
$$

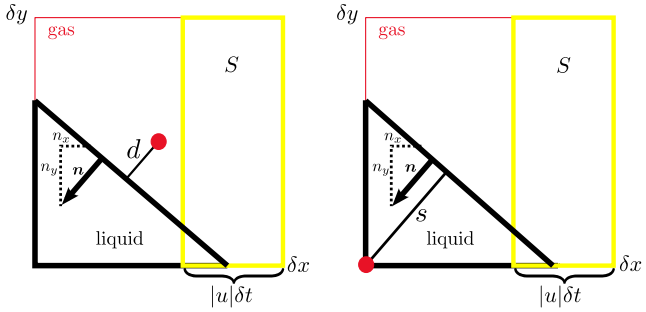
The distinction of cases in this equation is necessary due to our requirement that the region where we compute the liquid volume $\widetilde{\delta V_F}$ must contain the origin. Let us give an example in two dimensions. To this end, let $u > 0$ and go back to the 2D example illustrated in Figure 5.1. Recall that in Subsection 4.3.1 the number of interface configurations is reduced by reflecting the plane in such a way that all components of its unit normal $\boldsymbol{n} = (n_x, n_y, n_z)^T$ become negative *and* by shifting the plane's origin from the cell center to the lower left hand corner of the cell.

We define $S := [\delta x - |u|\delta t, \delta x] \times [0, \delta y]$ as illustrated in Figure 5.1. First, let $n_x \geq 0$ as shown in Figure 5.1(a). In this case, the origin s lies in the reflected image S_R of the subregion S after the computation of the reflected plane and the computation of the new position of the origin by (4.12). Then, we may compute the advected liquid volume directly in the subregion S_R, which in (5.2) amounts to $(\delta V_F)^{\rightarrow}_{i+1/2,j,k} = \widetilde{\delta V_F}(\boldsymbol{n}, s, |u|\,\delta t, \delta y, \delta z)$.

Now, let $n_x < 0$ as shown in Figure 5.1(b). Then, the origin s is not in the subregion S after the computation of the new position of the origin by (4.12) (here, no reflection is necessary). Then, we simply compute the liquid volume in the subregion $[0, \delta x - |u|\delta t] \times [0, \delta y]$ which *includes* the origin of s and subtract this volume from the overall liquid volume $F\delta x\delta y$ to obtain the liquid volume in S. Thus, we have $(\delta V_F)^{\rightarrow}_{i+1/2,j,k} = F\delta x\delta y\delta z - \widetilde{\delta V_F}(\boldsymbol{n}, s, \delta x - |u|\delta t, \delta y, \delta z)$ in (5.2).



(a) $n_x u \geq 0$



(b) $n_x u < 0$

Fig. 5.1: Placement of origin depending on the sign of $n_x u$. For the computation of the volume fractions, the origin of the plane needs to be in the the rectangular part S of the computational cell $\delta x \times \delta y \times \delta z$ as marked by yellow boxes. For $n_x u \geq 0$, the picture in the middle shows the reflected plane which is computed across the vertical dashed line. Then, the third picture shows the placement of the origin s defined by (4.12) in the liquid region as far from the interface as possible. Still, in this case the origin remains in the reflected image S_R of the subregion. If $n_x u < 0$, a reflection is unnecessary since $n_x < 0$ already, but the newly computed origin s remains outside the subregion S.

In summary, we are now able to compute the advected liquid volumes $(\delta V_F)^{\rightleftarrows}$ which are needed in (3.10) to obtain the liquid volume fractions $F_{i+1/2,j,k}$, $F_{i,j+1/2,k}$ and $F_{i,j,k+1/2}$ for the transport of F in (3.5)–(3.7).

6. Reinitialization of the LS function. The last step of our CLSVOF method is the reinitialization of the LS function with the help of the reconstructed interface, i.e. we want the LS function to be the exact signed normal distance to the reconstructed interface.

Before the reinitialization of the LS function and after the final reconstruction of the interface, the volume fractions are truncated to avoid unphysical flotsam and jetsam, which are generally created by round-off errors, i.e.

$$F_{i,j,k} = \begin{cases} 0 & \text{if } \phi_{i,j,k} < -\varepsilon \text{ or } F_{i,j,k} < 0 \\ 1 & \text{if } \phi_{i,j,k} > \varepsilon \text{ or } F_{i,j,k} > 1, \end{cases} \tag{6.1}$$

where ε denotes the thickness of the interface proportional to the mesh size h due

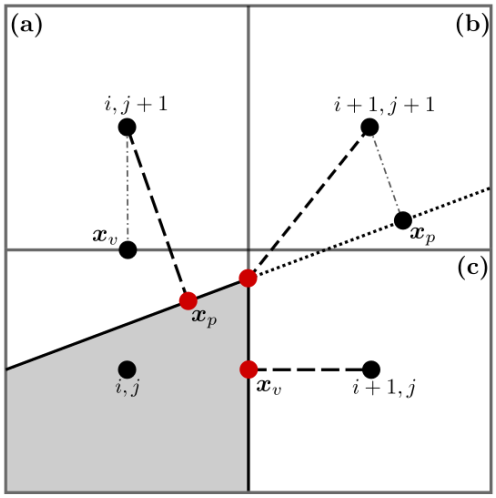


Fig. 6.1: Level-set redistancing: The gray area marks one phase and the white area marks the other one, while the thick black lines denote the reconstructed interface. The red points, which are connected to their cell centers by dashed lines, are those with the shortest distance to the interface. Possible candidates for these points are projection points $\boldsymbol{x}_p$ from the cell center onto the interface or points $\boldsymbol{x}_v$ on the boundary of cell (i, j, k); compare Table 6.1.

to smoothing of density and viscosity in this area. By restricting F to lie between 0 and 1, we introduce a truncation error and lose a very small amount of mass in the process. Note that this mass loss is typically an order of magnitude less than that lost in LS methods [21].

The reconstructed interface consists of piecewise linear plane segments as well as the plane segments at the cell boundaries that connect a plane in one cell with the plane in the cell next to it. The sign of the LS function can be determined by the sign of the VOF function $\operatorname{sgn}(F - 0.5)$. The magnitude of ϕ is calculated as the shortest distance from the cell center to any of the reconstructed interface segments. Thus, in the cell (i, j, k) with a reconstructed interface, we have to locate the point $\boldsymbol{x}_s$ on the interface which has the shortest distance to a neighboring cell center $\boldsymbol{x}'$ of a cell (i', j', k').

In summary the reinitialization algorithm consists mainly of the following two parts, which are illustrated in Figure 6.1 (a)–(c):

1. Try the simple cases: The point $\boldsymbol{x}_s$ can be found at the face center or face corner of cell (i, j, k) as in (c). Or, the projection of $\boldsymbol{x}'$ onto the interface lies in cell (i, j, k) as in (a).
2. Do the difficult cases: The point $\boldsymbol{x}_s$ is one of the vertices of the polygon in cell (i, j, k) as in (b). In 3D, $\boldsymbol{x}_s$ could also be the projection of $\boldsymbol{x}'$ onto one of the line segments of the polygon.

For the computation of $\boldsymbol{x}_s$ we have mainly consulted Sussman and Puckett [36] (2D/3D), Sussman [34] (2D/3D), Son and Hur [33] (2D), Son [32] (3D) and Wang et al. [39] (2D/3D). Most of the algorithms given in these articles describe the distance computation well for the simple interface configurations. However in three dimensions, the most difficult case is the one where we need to compute the vertices of the interface, which is not dealt with in detail in the works of Sussman [36] and [34]. Nevertheless, the technicalities of the vertex computation can be found in Son's article [32]. Then again, Wang et al. [39, p. 234] criticize Son's 3D algorithm for its complexity and describe their own new distance computation procedure in two and three dimensions. But while this procedure might work in 2D, we could not implement it successfully in 3D [17].

In the following, we propose a simple procedure for computing the interface ver-

Table 6.1: Description of points used in the reinitialization algorithm.

Point	Description
$\boldsymbol{x}'$	cell center of (i', j', k') within the narrow band about cell (i, j, k)
$\boldsymbol{x}_s$	the sought point on the interface in cell (i, j, k) with shortest distance to $\boldsymbol{x}'$
$\boldsymbol{x}_v$	point on the boundary of cell (i, j, k) with shortest distance to $\boldsymbol{x}'$, candidate for $\boldsymbol{x}_s$
$\boldsymbol{x}_p$	projection of $\boldsymbol{x}'$ onto the interface, candidate for $\boldsymbol{x}_s$

tices in 3D. In contrast to Son's approach, we do not rotate the interface to work with the reduced number of interface configurations, which would render the computation more complicated since we also need to transform the coordinates of the vertices back into the original system. Instead, we work directly in the original system with a larger amount of interface configurations and still compute the vertices efficiently by taking into account the distances of the *vertices of the computational cell* to the interface. In addition, we describe the more simple cases of the reinitialization procedure in detail by following the aforementioned articles and [39] in particular.

The following description is simplified by shifting the plane's origin to the origin of the computational domain $(0, 0, 0)$ so that from here on we define

$$\phi^R := n_x x + n_y y + n_z z + d' \tag{6.2}$$

with d' the distance from $(0, 0, 0)$ to the plane defined by (4.5). The *signed distance* of a an arbitrary point $\boldsymbol{p} = (p_x, p_y, p_z) \in \mathbb{R}^3$ to the above plane is defined by

$$\mathrm{D}(\boldsymbol{p}) := \boldsymbol{n} \cdot \boldsymbol{p} + d' = n_x p_x + n_y p_y + n_z p_z + d'. \tag{6.3}$$

This distance is positive, if $\boldsymbol{p}$ lies on the same side of the plane towards which the unit normal points and negative otherwise. *The projection of a point* $\boldsymbol{p} \in \mathbb{R}^3$ onto the plane is defined by

$$\mathrm{P}(\boldsymbol{p}) := \boldsymbol{p} - \mathrm{D}(\boldsymbol{p}) \cdot \boldsymbol{n}. \tag{6.4}$$

In what follows, we write the Euclidean distance between two points $\boldsymbol{p}^1$ and $\boldsymbol{p}^2$ as

$$\mathrm{d}_{\mathrm{euc}}(\boldsymbol{p}^1, \boldsymbol{p}^2) := \sqrt{(p_x^2 - p_x^1)^2 + (p_y^2 - p_y^1)^2 + (p_z^2 - p_z^1)^2}.$$

Furthermore, we introduce an auxiliary level-set function ϕ^{aux}, which we initialize in all computational cells with a very large value. This function is needed since we aim at the minimum distance of each cell center to the reconstructed interface. At the end of our algorithm, we set $\phi = \phi^{\mathrm{aux}}$ in all considered cells.

Now, let (i, j, k) be a given cell with a reconstructed interface as described in Subsection 4.1. Then, (i', j', k') with $|i' - i| \leq K$, $|j' - j| \leq K$ and $|k' - k| \leq K$ are the cells within a narrow band of width K about (i, j, k). Let us denote the cell center of (i', j', k') by $\boldsymbol{x}'$ and the point on the interface with the shortest distance to $\boldsymbol{x}'$ by $\boldsymbol{x}_s$. In Table 6.1, we give an overview and description of all points used in the reinitialization algorithm.

1. If (i, j, k) is a cell with a reconstructed interface, the value of the auxiliary function ϕ^{aux} simply becomes the distance of the cell center to the reconstructed interface, i.e.

$$\phi^{\mathrm{aux}}_{i,j,k} = \mathrm{sgn}(F_{i,j,k} - 0.5) |\,\mathrm{D}(\boldsymbol{x}')| = \mathrm{sgn}(F - 0.5)\, |n_x x' + n_y y' + n_z z' + d'|\,,$$

where d' is now the distance of the computational domain's origin to the plane (and no longer from the corner or center of the computational cell).

2. As a candidate for $\boldsymbol{x}_s$, determine the point $\boldsymbol{x}_v$ on the boundary of cell (i,j,k) with the shortest distance to the cell center of (i',j',k'). This point will always be found at the corner of a cell face of (i,j,k) or at the center of a cell face of (i,j,k). Its coordinates are given by

$$\boldsymbol{x}_v := (x_{i+\frac{l}{2}}, y_{j+\frac{m}{2}}, z_{k+\frac{n}{2}})$$

with[1]

$$\begin{aligned} l &= \max(-1, \min(1, i'-i)) \\ m &= \max(-1, \min(1, j'-j)) \\ n &= \max(-1, \min(1, k'-k)). \end{aligned}$$

Then, we compute the signed distance of the point $\boldsymbol{x}_v$ to the reconstructed interface by

$$\mathrm{D}(\boldsymbol{x}_v) = d' + n_x x_v + n_y y_v + n_z z_v.$$

This distance is positive if the interface's normal vector points to the same side of the interface, where $\boldsymbol{x}_v$ lies. If the sign of this distance is different from the sign of the LS function in the cell center $\boldsymbol{x}'$ of cell (i',j',k'), i.e. $\mathrm{D}(\boldsymbol{x}_v)\,\mathrm{sgn}(\phi_{i',j',k'}) \leq 0$, we know that $\boldsymbol{x}_v$ and $\boldsymbol{x}'$ belong to different phases (Fig. 6.1 (c)). Then, we update ϕ^{aux} by

$$\phi^{\mathrm{aux}}_{i',j',k'} = \mathrm{sgn}(F_{i',j',k'} - 0.5) \min\left(\mathrm{d}_{\mathrm{euc}}(\boldsymbol{x}_v, \boldsymbol{x}'), |\phi^{\mathrm{aux}}_{i',j',k'}|\right)$$

with $\mathrm{d}_{\mathrm{euc}}(\cdot,\cdot)$ the Euclidean distance between these points defined by (6). Instead, if $\mathrm{D}(\boldsymbol{x}_v)\,\mathrm{sgn}(\phi_{i',j',k'}) > 0$, both points $\boldsymbol{x}_v$ and $\boldsymbol{x}'$ belong to the same phase; see Figure 6.1(a)), and we proceed with the next step.

3. Then, the next likely candidate for $\boldsymbol{x}_s$, is the projection of $\boldsymbol{x}'$ onto the interface, i.e. the point

$$\boldsymbol{x}_p = \mathrm{P}(\boldsymbol{x}') \overbrace{:=}^{(6.4)} \boldsymbol{x}' - \mathrm{D}(\boldsymbol{x}')\boldsymbol{n}.$$

If $\boldsymbol{x}_p$ is inside cell (i,j,k), this point must lie directly on the interface and, by definition, is the one with the smallest distance to $\boldsymbol{x}'$. In this case we set

$$\phi^{\mathrm{aux}}_{i',j',k'} = \mathrm{sgn}(F_{i',j',k'} - 0.5) \min\left(\mathrm{d}_{\mathrm{euc}}(\boldsymbol{x}_p, \boldsymbol{x}'), |\phi^{\mathrm{aux}}_{i',j',k'}|\right).$$

If the projection point is not in cell (i,j,k), we proceed with the next step.

4. With the above part of the algorithm all the simple standard cases are dealt with. Indeed, we now know that $\boldsymbol{x}_s$ must lie on one of the intersection points of the plane in cell (i,j,k) with the boundary of cell (i,j,k) (i.e. on one of the vertices of the at

[1] This definition of $\boldsymbol{x}_v$ can be understood as follows (cf. Fig. 6.1): If (i',j',k') is a direct neighbor of (i,j,k) (e.g. $i' = i+1, j = j, k = k$), then $l = \max(-1, \min(1, i+1-i)) = \max(-1,1) = 1$, $m = \max(-1, \min(1, j-j)) = 0$ and $n = 0$ so that $\boldsymbol{x}_v = x_{i+\frac{1}{2},j,k}$, which is a face center of cell (i,j,k). If (i',j',k') is a diagonal neighbor (e.g. $i' = i+1, j = j+1, k = k$), then $l = 1$, $j = 1$, $k = 0$ and $\boldsymbol{x}_v = x_{i+\frac{1}{2},j+\frac{1}{2},k}$, which is a face corner.

most 6-sided polygon; cf. Figure 6.2). Additionally, we have to take into account that $\boldsymbol{x}_s$ could be the projection point of a neighboring cell onto the side of the polygon.

In order to find $\boldsymbol{x}_s$, we propose the following algorithm: Set $\min_{\text{dist}}$ to a very large value and consider the vertices of the computational cell which are connected by 12 edges. For these edges $s = 1, ..., 12$:

(a) Consider the pair of vertices $(\boldsymbol{p}_{s_1}, \boldsymbol{p}_{s_2})$ of the computational cell which are connected by this edge.

(b) Compute the distances $\mathrm{D}(\boldsymbol{p}_{s_1})$ and $\mathrm{D}(\boldsymbol{p}_{s_2})$ of these points to the interface.[2]

(c) If $\mathrm{D}(\boldsymbol{p}_{s_1})\,\mathrm{D}(\boldsymbol{p}_{s_2}) > 0$, the interface does not cut the edge (since both points lie on the same side of the interface). In this case, go back to (a) and consider the next edge. Otherwise, the interface does cut the edge and we proceed with the next step.

(d) Compute the intersection point of the interface with the edge which we denote by $\boldsymbol{p}^{\text{is}}_{s_1 s_2}$. Set $\min_{\text{dist}} = \min\left(\mathrm{d}_{\text{euc}}\left(\boldsymbol{p}^{\text{is}}_{s_1 s_2}, \boldsymbol{x'}\right), \min_{\text{dist}}\right)$.

(e) Compute the projection of $\boldsymbol{x'}$ onto both of the polygon's interface segments which have $\boldsymbol{p}^{\text{is}}_{s_1 s_2}$ as a common vertex. We denote the projection points by $\boldsymbol{p}^{\text{pr}_1}_{s_1 s_2}$ and $\boldsymbol{p}^{\text{pr}_2}_{s_1 s_2}$. If the point $\boldsymbol{p}^{\text{pr}_1}_{s_1 s_2}$ is in cell (i, j, k), we set $\min_{\text{dist}} = \min\left(\mathrm{d}_{\text{euc}}\left(\boldsymbol{p}^{\text{pr}_1}_{s_1 s_2}, \boldsymbol{x'}\right), \min_{\text{dist}}\right)$. If the point $\boldsymbol{p}^{\text{pr}_2}_{s_1 s_2}$ is in cell (i, j, k), set $\min_{\text{dist}} = \min\left(\mathrm{d}_{\text{euc}}\left(\boldsymbol{p}^{\text{pr}_2}_{s_1 s_2}, \boldsymbol{x'}\right), \min_{\text{dist}}\right)$.

(f) Go to (a) and consider the next edge. If all 12 edges have been taken care of, go to the last step.

(g) Set $\phi^{\text{aux}}_{i',j',k'} = \mathrm{sgn}(F_{i',j',k'} - 0.5) \min\left(|\min_{\text{dist}}|, |\phi^{\text{aux}}_{i',j',k'}|\right)$

5. Finally, set $\phi_{i,j,k} = \phi^{\text{aux}}_{i',j',k'}$ in all cells where ϕ^{aux} now contains meaningful values inside the narrow band of K cells. All other cells outside the narrow band can be filled with a constant: For example, on nearly uniform grids we compute the maximum diagonal length $d_{\max}$ of all computational cells. In cells which lie outside the narrow band of K cells, we set

$$\phi_{i,j,k} = \begin{cases} -K d_{\max} - d_{\max} & \text{if } \phi_{i,j,k} < 0 \\ K d_{\max} + d_{\max} & \text{if } \phi_{i,j,k} > 0, \end{cases} \tag{6.5}$$

i.e. these cells are filled with a constant distance to the interface.

What makes the algorithm described in step 4 fast is that by comparing the sign of the distance of the cell's vertices to the interface, we know very quickly which edges of the computational cell intersect the polygon. Then, since the coordinates of the vertices of the edges are already stored as grid information, we can compute the vertices of the polygon and the projection points onto the line segments very efficiently.

Let us exemplify their computation for a pair of vertices $\boldsymbol{p}_{11} = (p_1, p_2, p_3)$ and $\boldsymbol{p}_{12} = (p_1 + \delta x, p_2, p_3)$ whose edge intersects the polygon (cf. Fig. 6.2). Then, $\boldsymbol{p}^{\text{is}}$ can simply be computed as

$$\boldsymbol{p}^{\text{is}} = \left(\frac{n_y p_2 + n_z p_3 + d'}{-n_x},\, p_2,\, p_3\right).$$

Furthermore, we know that the polygon segments which intersect this vertex must lie

[2] See equation (6.3) for the definition of the *signed distance* of a point $\boldsymbol{p}$ to a plane. This distance is positive, if $\boldsymbol{p}$ lies on the same side of the plane towards which the unit normal points and negative otherwise.

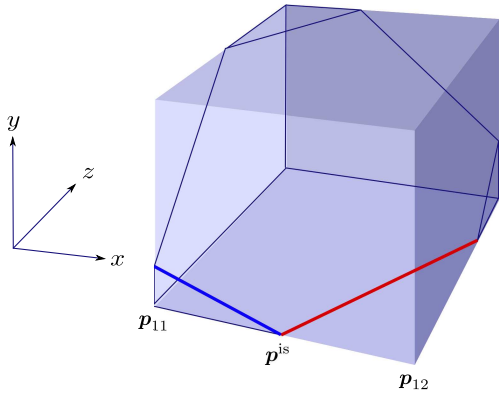


Fig. 6.2: Illustration for finding the vertices of the polygon and the projection points onto the segments of the polygon. The vertices $\boldsymbol{p}_{11} = (p_1, p_2, p_3)$ and $\boldsymbol{p}_{12} = (p_1 + \delta x, p_2, p_3)$ of the computational cell are connected by the same edge which is intersected by the 6-sided polygon. This intersection point is denoted by $\boldsymbol{p}^{\text{is}}$. The segments of the polygon which are connected to $\boldsymbol{p}^{\text{is}}$ lie in the $y = p_2$-face. (thick red line) and the $z = p_3$-face (thick blue line) of the cell.

within the faces $y = p_2$ and $z = p_3$ of the computational cell.[3] They obey the linear equations

$$n_x x + n_z z + (d' + n_y p_2) = 0 \text{ and} \tag{6.6}$$
$$n_x x + n_y y + (d' + n_z p_3) = 0. \tag{6.7}$$

We normalize these equations and obtain the new parameters

$$\tilde{n}_x := n_x/\sqrt{n_x^2 + n_y^2}, \qquad \tilde{n}_z := n_z/\sqrt{n_x^2 + n_y^2}, \qquad \tilde{d} := (d' + n_y p_2)/\sqrt{n_x^2 + n_y^2}$$

for the first polygon segment and

$$\hat{n}_x := n_x/\sqrt{n_x^2 + n_z^2}, \qquad \hat{n}_y := n_y/\sqrt{n_x^2 + n_z^2}, \qquad \hat{d} := (d' + n_z p_3)/\sqrt{n_x^2 + n_z^2}.$$

for the second polygon segment. Then equations (6.6) and (6.7) become

$$\tilde{n}_x x + \tilde{n}_z z + \tilde{d} = 0 \text{ and} \tag{6.8}$$
$$\hat{n}_x x + \hat{n}_y y + \hat{d} = 0. \tag{6.9}$$

In the next step, we compute the projection of $\boldsymbol{x}'$ onto both of the polygon's interface segments (6.8) and (6.9). Since we know that these segments lie in the faces $y = p_2$ and $z = p_3$, respectively, these entries of the coordinates of the projection point are fixed, while the remaining two entries can be computed by the standard projection onto the normalized line equations (6.8) and (6.9). Thus,

$$\boldsymbol{p}^{\text{pr}_1} = \left(x_1' - (\tilde{n}_x x_1' + \tilde{n}_z x_3' + \tilde{d})\tilde{n}_x,\ p_2,\ x_3' - (\tilde{n}_x x_1' + \tilde{n}_z x_3' + \tilde{d})\tilde{n}_z\right) \text{ and}$$
$$\boldsymbol{p}^{\text{pr}_2} = \left(x_1' - (\hat{n}_x x_1' + \hat{n}_y x_2' + \hat{d})\hat{n}_x,\ x_2' - (\hat{n}_x x_1' + \hat{n}_y x_2' + \hat{d})\hat{n}_y,\ p_3\right),$$

which concludes our example for the computation of $\boldsymbol{x}_s$ as one one of the vertices of the at most 6-sided polygon or as the projection of a neighboring cell center onto the side of the polygon.

[3] Here, we define the x_f-face of cell (i, j, k) as

$$\left\{(x, y, z) :\ x = x_f,\ y_{j-1/2} \le y \le y_{j+1/2},\ z_{k-1/2} \le z \le z_{k+1/2}\right\},$$

and we define the other faces analogously.

Finally note that all parts of the implementation of the CLSVOF method have to be done with special care concerning floating point comparisons, which arise in the computation of the volume fractions and within the reinitialization equation. Additionally, all divisions have to be carefully controlled since the denominators can become tiny. Although these considerations hold for every computer program, the sheer number of such occurrences within the implementation of the CLSVOF method necessitate this remark.

7. Parallelization. Our flow solver NaSt3DGPF [24] already works completely in parallel and employs a natural parallelization strategy for the Navier-Stokes equations: the Cartesian grid Ω^h is decomposed into Q overlapping subdomains Ω_q^h, $q = 1, \ldots, Q$, which are in turn treated by one of the Q processors. Furthermore, we define the domain Ω_{q+}^h which equals the domain Ω_q^h plus an additional strip of ghost cells in each space direction. Then, individual processes no longer require access to the entire data structure and the solution of iterative algorithms can be distributed among them. Furthermore, we assure the convergence of the parallelized algorithm by an exchange of relevant data between processes treating adjacent subdomains (neighboring processes). Each of these subdomains is extended by an artificial boundary, which guarantees the well-definedness of the equations on every process. The details of our parallelization approach are extensively described in [4, 16, 17].

In this section, we focus on the parallelization of the CLSVOF method, which requires additional communication between processes on adjacent subdomains. To our knowledge, the parallelization of the reinitialization procedure is described nowhere else in the literature.

Let us focus on the transport part of our algorithm. In the transport steps (3.5), (3.6) and (3.7) we not only need boundary values for ϕ and F but also boundary values for the interface normals n_x, n_y n_z and the interface's distance to the origin, which are all computed in the interface reconstruction process. These values need to be communicated if they lie outside the domain Ω_{q+}^h on which we compute the interface reconstruction.

The most challenging part of the parallelization of the CLSVOF method is the reinitialization, in which the LS function is computed as the exact distance of the cell centers to the reconstructed interface. Here, (i, j, k) is a given cell with a reconstructed interface as described in Subsection 4.1, while (i', j', k') with $|i' - i| \leq K$, $|j' - j| \leq K$ and $|k' - k| \leq K$ are the cells within a narrow band of width K about (i, j, k). Again, we denote the cell center of (i', j', k') by $\boldsymbol{x}'$ and the point on the interface with the shortest distance to $\boldsymbol{x}'$ by $\boldsymbol{x}_s$ (cf. Table 6.1). Then, in each cell center the LS value is the minimum distance to all considered interface segments, i.e.

$$\phi_{i',j',k'} := \min_{i,j,k}(\mathrm{d}_{\mathrm{euc}}(\boldsymbol{x}'_{i',j',k'}, \boldsymbol{x}^s_{i,j,k})), \tag{7.1}$$

for $(i - i') \geq K$, $(j - j') \geq K$ and $(k - k') \geq K$. Here, $\mathrm{d}_{\mathrm{euc}}(\boldsymbol{x}'_{i',j',k'}, \boldsymbol{x}^s_{i,j,k})$ is the Euclidean distance of the cell center $\boldsymbol{x}'_{i',j',k'}$ to each interface segment, i.e. $\boldsymbol{x}^s_{i,j,k}$ is the point on the interface with shortest distance to $\boldsymbol{x}'_{i',j',k'}$.

There are two strategies for the parallelization of the reinitialization in the CLSVOF method. The first and seemingly more natural strategy is for each process to check only its own domain Ω_q^h for cells with an interface. Then, the distances from the cell centers of neighboring processes have to be computed and communicated. The second strategy is for each process to check its own domain and the K cells of neighboring processes for interface segments (requiring communication). Then, each process can

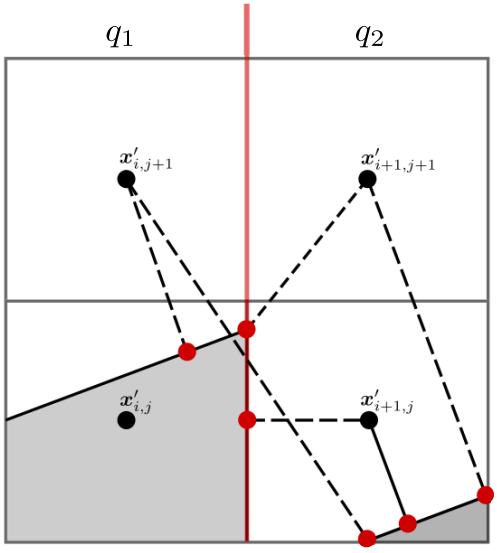


Fig. 7.1: Example for the parallelization of the reinitialization within the CLSVOF method. The red line denotes the boundary between the processes q_1 and q_2. All cell centers are marked by black dots, and the distance from the cell centers to the interface segments are drawn by dashed black lines. Then, the red dot denotes the point with the shortest distance from the respective cell center to the interface segment.

compute the distances to these segments in its own domain Ω_q^h, for which no communication is necessary.

Here, we apply the second strategy, i.e. each process checks its own domain and the K cells of neighboring processes for interface segments and computes distances to these interfaces in its own domain Ω_q^h only. The first part of our strategy requires communication among the processes since for the K strips of boundary cells of Ω_q^h each process needs to know

1. if there is a cell containing an interface,
2. the three values of its normal and its distance to the origin,
3. the value of ϕ in each cell,
4. and the value of F in each cell.

This is a standard communication step, which is more costly due to the larger number of functions and strips of boundary cells that have to be considered. However, it has to be done only once for every time step and can be further reduced, if the position of the interface is taken into account. Furthermore, the minimum distance can be computed by each process individually, which renders our strategy less complex.

Let us further exemplify our approach by the 2D example in Figure 7.1. Here, q_1 computes the minimum of the distances from $\boldsymbol{x}'_{i,j+1}$ to both interface segments in cell (i,j) and $(i+1,j)$, while q_2 computes the distance to these interface segments from $\boldsymbol{x}'_{i+1,j+1}$. Furthermore, both q_1 and q_2 know about the interface segments in cells (i,j) and $(i+1,j)$, so that no unnecessary computations are performed. Thus, each process performs the minimum computation (7.1) with the restriction that $\boldsymbol{x}'_{i',j'} \in \Omega_q^h$. Each process can do this computation individually and no further communication is necessary.

The details of our implementation are as follows:

1. Each process checks its own domain for interface segments as well as the K strips of boundary cells of Ω_q^h. If the process reaches the boundary of the overall computational domain Ω^h, we only check its first layer of boundary cells, not all three rows of ghost cells.
2. If a cell with a reconstructed interface is found, each process only computes

Algorithm 1: Nesting of loops for the parallelization of the reinitialization equation.

```
for(i=il-K; i<=iu+K; i++)
for(j=jl-K; j<=ju+K; j++)
for(k=kl-K; k<=ku+K; k++){
  if(i>=0 && i<=imax+1 && j>=0 && j<=jmax+1 && k>=0 && k<=kmax+1)
  if(IsCellWithReconstructedInterface(i, j, k)){
      for(p=pl; p<=pu; p++)
      for(q=ql; q<=qu; q++)
      for(r=rl; r<=ru; r++)
      if(!IsCellWithReconstructedInterface(p, q, r)){
      \\find minimal distance of point in cell p,q,r
      \\to interface in i,j,k
      }
  }
}
```

the minimal distance to the interface in its own domain Ω_q^h and in the ghost cells rows of Ω^h.

We have

$$\Omega^h := [i_{\min}, i_{\max}] \times [j_{\min}, j_{\max}] \times [k_{\min}, k_{\max}] \text{ and } \Omega_q^h := [i_l, i_u] \times [j_l, j_u] \times [k_l, k_u]$$

without ghost cells. Then, in the x-direction, we define the lower and upper boundaries in which each process computes distances by

$$p_l := \begin{cases} i_{\min} - 3 & \text{if } i - K < i_{\min} - 1, \text{ i.e. we are below the lower boundary of } \Omega^h \\ i_l & \text{if } i - K < i_l, \text{ i.e. we are at the lower boundary of } \Omega_q^h \\ i - K & \text{else,} \end{cases} \tag{7.2}$$

and

$$p_u := \begin{cases} i_{\max} + 3 & \text{if } i + K > i_{\max} + 1, \text{ i.e. we are above the upper boundary of } \Omega^h \\ i_u & \text{if } i + K > i_u, \text{ i.e. we are at the upper boundary of } \Omega_q^h \\ i + K & \text{else,} \end{cases} \tag{7.3}$$

and q_u, q_o, r_u and r_o for the indices j and k are defined analogously. Note that the '-3' or '+3' indicates that we have three rows of ghost cells for the boundary of Ω^h. If more or less rows are needed, this number has to be adapted accordingly.

The nesting of loops for the parallelization of the reinitialization equation is then given in Algorithm 1. Thus, our first three loops make sure that each processor checks its own domain for cells which contain an interface as well as the K boundary cells. Then, the 'if statement' lets each process consider one strip of boundary cells of Ω^h only. If a cell with a reconstructed interface is found, each process looks at the K neighboring cells and computes their distance to the interface cell. Additionally, the next three loops and the conditions for their lower (7.2) and upper bound (7.3) make sure that each process only computes distances its own domain Ω_q^h.

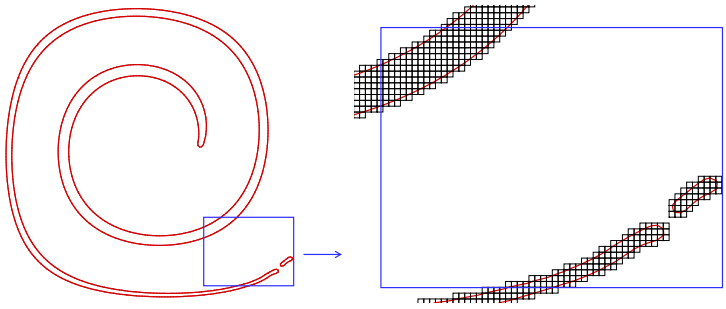


Fig. 8.1: Circle at maximum stretching. On the left, the contour computed by the CLSVOF method is shown. The blue rectangle is magnified on the right, where grid cells are drawn. The very thin filaments are resolved by 2–3 grid cells only.

8. Numerical Results. In this section, we are concerned with two standard test cases with prescribed velocity field. First, we solve the stretching of a circle in a shear velocity field as proposed by Rider and Kothe [30], which is a two-dimensional example. Our second advection case is the fully three-dimensional deformation of a sphere. Both of these test cases allow for a direct comparison between the different interface capturing methods, without involvement of the flow solver and without the costly solution of the Poisson equation.

Our parallel simulations were computed on the *Atacama* cluster of the Institute for Numerical Simulation [13], Bonn University. This cluster consists of 78 Dell PowerEdge M620 nodes with 1264 cores and has a Linpack performance of 20630 GFlop/s, while maintaining a total memory of 4992 GB. Note that all surface and volume integrals for the analysis of results are computed by the ParaView [26] or VisIt software [2] in a post-processing step. Both software tools are able to reconstruct the iso 0 level-set surface. Then, an integration over the interface Γ_f can be done without the use of the δ-function, which always comes with a thickness of several cells across the interface and introduces a constant error in the integration. Here, ParaView and VisIt rely on the Marching Cubes algorithm [20] for isosurface extraction. For details on the applied filters; see [17]. Additionally, for visualization purposes only, we use the Tecplot 360 software (Release 1, 2013) [38].

8.1. Two-dimensional stretching of a circle in a shear velocity field. Our first test case is the two-dimensional stretching of a circle in a shear velocity field as proposed by Rider and Kothe [30]. This advection test is a realistic problem since interfaces undergo strong topological changes, including merging and fragmentation. Our presented test takes such shear effects into account and is further challenging due to severe topological changes with very thin filaments on the scale of the mesh size (Fig. 8.1). The LS and the CLSVOF method are evaluated by their ability to cope with these changes and their ability to preserve the initial mass of the sphere. In addition, we are interested in the computational efficiency of both methods.

8.1.1. Setup of the numerical experiment. We choose a flow domain Ω with size $\Omega = [0, 1.0] \times [0, 1.0] \times [0, 0.5]$ which is quasi-periodic with respect to the z-axis. A cylinder with radius 0.15 is centered at $\bar{\boldsymbol{x}}(t = 0) = (0.5, 0.75, 0.25)^{\mathsf{T}}$. In the following, we denote the area occupied by the cylinder by $\Omega_1 = \Omega_1(t) \subset \Omega$. Due to the quasi-periodic setup, we then have a circle on each slice in the periodic z-direction. Hence,

Table 8.1: Simulation parameters for the two-dimensional stretching of a circle in a shear velocity field.

u:	$-\sin^2(\pi x)\sin(2\pi y)\cos(\pi t/T)$
v:	$\sin^2(\pi y)\sin(2\pi x)\cos(\pi t/T)$
w:	0
final time:	$T = 8$
flow domain:	$\Omega = [0, 1.0] \times [0, 1.0] \times [0, 0.5]$
circle radius:	0.15
circle center:	$(0.5, 0.75, 0.25)$
interface thickness:	$\epsilon = 1.75h$
grid resolution:	$128 \times 128 \times 5$ and $256 \times 256 \times 10$
quasi 2D:	periodic boundary conditions in z-direction
boundary conditions:	no-slip except for z-direction

in the following, we always use the term 'circle' – although we are talking about a cylinder from the three-dimensional point of view.

The prescribed shearing field is given by

$$\begin{aligned} u &= -\sin^2(\pi x)\sin(2\pi y)\cos(\pi t/T) \\ v &= \sin^2(\pi y)\sin(2\pi x)\cos(\pi t/T) \\ w &= 0 \end{aligned} \tag{8.1}$$

with $T = 8$ the time when the flow returns to its initial position. Due to the shearing field, the circle deforms to a vortex with maximum stretching at $t = 4$. Then the velocity field is inverted and the vortex is compressed back to its initial position and circular shape. Note that a multiplication by $\cos(\pi t/T)$ causes any incompressible flow field to return to its initial state at $t = T$ [18, 30], a feature employed in this section as well as for the three-dimensional deformation field in the next section. Then, the differences in data at $t = 0$ and $t = T$ can be used for error measurements. All numerically and physically relevant parameters are summarized in Table 8.1.

We compute the solution on two grids with 128×128 and 256×256 grid cells in the x- and y-direction. Since we are aiming at a quasi-2D solution with our 3D Navier-Stokes solver, we resolve the domain of length 0.5 in z-direction by 5 or 10 grid cells for the lower and higher resolution, respectively.[4] We then have periodic boundary conditions in the z-direction and no-slip boundary conditions hold at all remaining boundary faces. On both grids the time step is restricted by

$$\delta t = \mathrm{cfl} \cdot \frac{h_{\min}}{\max_{(i,j,k)} \left(u_{i,j,k}, v_{i,j,k}, w_{i,j,k}\right)} \tag{8.2}$$

with CFL number $\mathrm{cfl} = 0.25$ and $h_{\min}$ the smallest mesh size.

Our simulations are conducted in parallel on one 16-core node of the *Atacama* cluster. Since the velocity field is prescribed, we do not solve the full Navier-Stokes equations. We only have to concern ourselves with the solution of the transport equation and with the reinitialization of the LS function.

[4] Normally, we would not apply two different resolutions in the periodic direction. However, the CLSVOF code cannot yet deal with anisotropic grids which is due to our present implementation – not the method itself.

8.1.2. Expected behavior. Certainly, for this test, we expect better mass conservation with the CLSVOF method compared to our LS method at least on coarse grids. If we choose a fine enough grid, both the LS method and the CLSVOF method should be able to conserve a satisfactory amount of mass and to recover the initial circular configuration.

Concerning the computing time, we expect both the CLSVOF and the LS method to behave similarly. The bottleneck for the LS method is the reinitialization of the LS function, which takes up most computing time. In the CLSVOF method, the LS function ϕ^{n+1} needs to be exact within a narrow band of K cells about the interface, where we usually set $K = 4$. With N the number of cells which contain an interface reconstruction, the speed of the algorithm is $\mathcal{O}(K^3N)$ [36], which is also that of the reinitialization of the LS method in a tube of K cells about the zero level-set. In addition to the LS method, we have to transport the VOF function and to compute the reconstruction of the interface for the CLSVOF method. Instead, for the LS method, we employ several costly features in the reinitialization equation like a high-order WENO method in space and a Runge-Kutta scheme for time discretization. All in all, we therefore expect a similar computational efficiency of both methods.

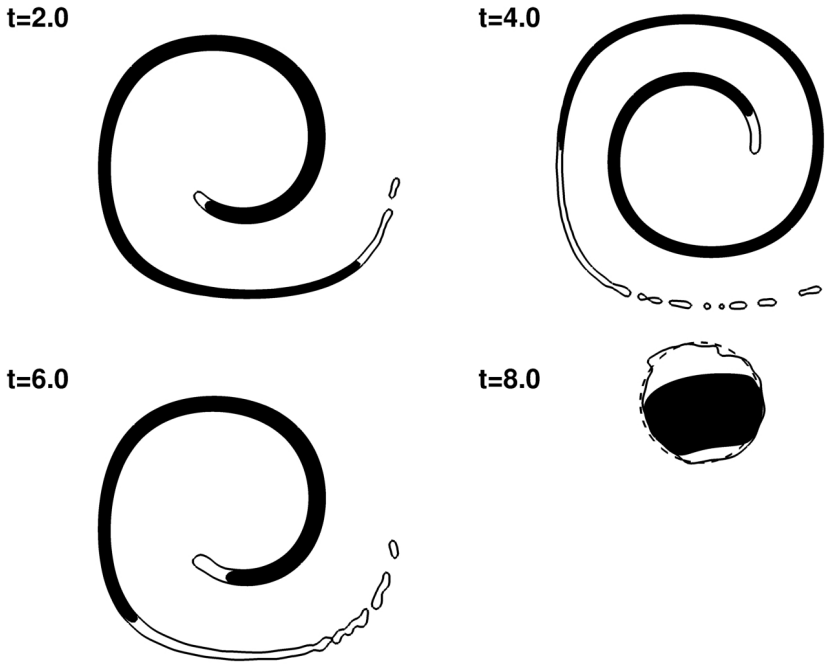
8.1.3. Discussion of results. The development of the circle in the shear flow over time is shown in Figure 8.2, where we draw the zero level-set contour extracted with the Tecplot software on the $z = 0.25$-slice at times $t = 2$, 4, 6 and 8. The top four figures refer to the coarse grid solution ($h = 1/128$), while the four figures on the bottom refer to the fine grid solution ($h = 1/256$). At $t = 8$, we also plot the initial circle as a dashed black line, which should be recovered by the respective numerical methods.

In Figure 8.2, the CLSVOF interface is the black line filled with white, while the LS solution is marked by solid black. As expected, the CLSVOF method is much better at mass conservation than the LS method. Note that the interface of the LS method is up to a certain point in time always almost equal to that recovered by the CLSVOF method. But the thin filaments which develop at the front or back of the stretched circle are simply lost with the LS method.

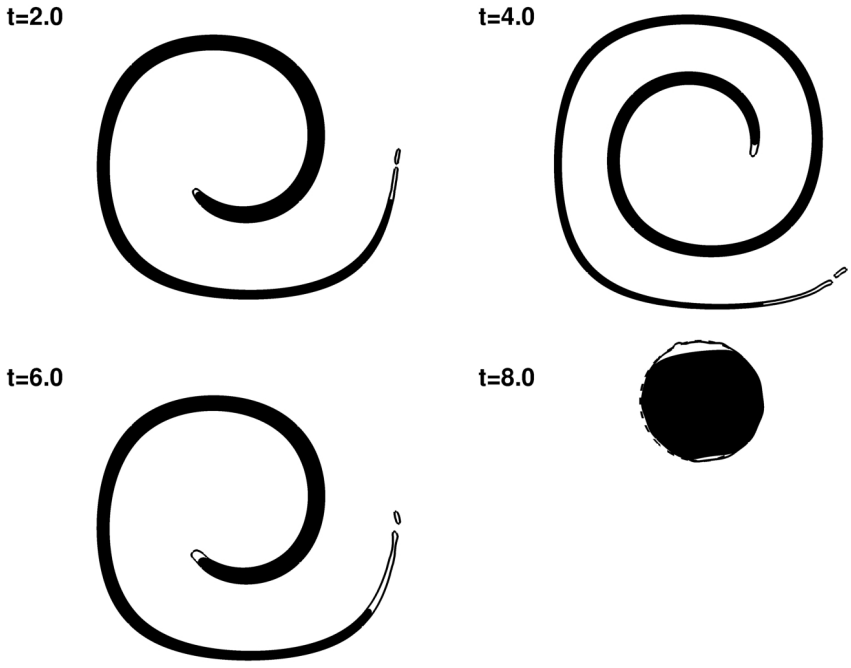
The mass loss is most notable on the coarse grid with $h = 1/128$ (Fig. 8.2(a)). At maximum stretching ($t = 4$), there is a vast difference between the length of the vortex with the LS and the CLSVOF method. Several satellite droplets evolve at the rear end of the vortex with the CLSVOF method, which tries to recover the thin filaments. After the flow reverses, these droplets become an irregular structure, which can be seen at $t = 6$. Then, the initial circle, which is drawn as a dashed black line in the last picture, cannot be recovered exactly, but the CLSVOF method is still very close. In contrast, due to the vast amount of mass lost with the LS method, the LS solution is nowhere near spherical in the last frame of Figure 8.2(a).

As expected, the differences between the CLSVOF and the LS method are less prominent on the finer grid (Fig. 8.2(b)). Note that on this fine mesh, the filament at the rear end of the vortex is resolved by only 2–3 grid cells (cf. Figure 8.1). The LS solution now loses much less mass than on the coarse grid but still performs less well than the CLSVOF method. Additionally, only one satellite drop evolves on the finer grid at $t = 4$, so that the CLSVOF method recovers the circle even more accurately than on the coarse grid. Again, the final shape produced by the LS method is less circular due to the mass loss over time.

In a last step, we measure the computing time of both methods for both grid resolutions. All simulations were conducted in parallel on one 16-core node of the



(a) $h = 1/128$



(b) $h = 1/256$

Fig. 8.2: Two-dimensional stretching of a circle in a shear velocity field for two different grid resolutions on the slice $z = 0.25$. The outer lines mark the zero level-set and are filled with black for the LS method and with white for the CLSVOF method. Quite noticeably, the LS solution always remains within the bounds of the CLSVOF solution. For comparison, we draw the initial circle (dashed black) in the $t = 8$ frame for both grid resolutions.

Table 8.2: Computing time for the two-dimensional stretching of a circle in a shear velocity field on one 16-core node of the *Atacama* cluster.

	LS	CLSVOF
1/128	0 h 33 min	0 h 19 min
1/256	6 h 10 min	2 h 56 min

Atacama cluster. The resulting computing time is shown in Table 8.2. The CLSVOF method performs admirably and is twice as fast as the LS method on the finest grid. This might be due to the very expensive components of the LS reinitialization scheme such as the fifth order WENO scheme in space and the third-order Runge-Kutta scheme in time, which yield an accurate but computationally expensive solution.

Next, we present some exemplary results from the literature, where the test case of the 'circle in shear flow' has already been solved with the CLSVOF method.

8.1.4. Comparison to results from the literature. Our test case has been studied with the CLSVOF method by Ménard, Tanguy and Berlemont in [23] and by Wang, Yang and Stern in [40] (cf. also [39]). In [23] the CLSVOF method is specifically adapted to handle very thin filaments for the simulation of the primary break-up of liquid jets. This is achieved by a modification of the interface reconstruction stencil (cf. Section 4) when there is more than one interface front in the stencil domain, i.e. when there are very fine filaments of one fluid phase. Due to this modification, very good results are obtained at $t = \frac{T}{2}$ when the flow is inverted as well as for the final configuration $t = T$ (with $T = 6$ in this work) compared to the exact Lagrangian solution.

The results presented in [40] are obtained by a CLSVOF method which uses a Lagrangian method with a second-order Runge-Kutta scheme for the advection of the piecewise linear interface. The results obtained in that work are comparable to our own results. From a merely qualitative point of view, the circle seems to be recovered slightly better in [40] on the coarse grid with $h = 1/128$, but results with a finer mesh resolution are missing.

8.1.5. Remarks. We can draw several conclusions from this challenging test case.

First, the CLSVOF is well equipped to deal with severe topological changes and to maintain the mass of thin filaments developing on the scale of the mesh size. Furthermore, the circle is quite accurately returned to its initial position. Remarkably, the results with the CLSVOF and the LS method are very close to each other: Up to the onset of mass loss, the interface of the LS method is always almost equal to that recovered by the CLSVOF method. This is a very good result, since the re-distance algorithm in the CLSVOF method is completely different from the classical Hamilton-Jacobi reinitialization in the LS method. Therefore, we can infer that our new vertex finding algorithm in the re-distancing of the CLSVOF method works perfectly. Last, the CLSVOF method is least computationally expensive, which is indispensable for the computation of highly resolved real life problems such as droplet impact.

Second, the results with the LS method are not too far from those with the CLSVOF method on the finest grid. Actually, this is what we expect of the LS method, which exhibits several high-performance features like WENO schemes for reinitialization and transport and high-order time discretization methods. These methods, however, have

Table 8.3: Simulation parameters for the three-dimensional deformation of a sphere.

u:	$+2\sin^2(\pi x)\sin(2\pi y)\sin(2\pi z)\cos(\pi t/T)$
v:	$-\sin^2(\pi y)\sin(2\pi x)\sin(2\pi z)\cos(\pi t/T)$
w:	$-\sin^2(\pi z)\sin(2\pi x)\sin(2\pi y)\cos(\pi t/T)$
final time :	$T = 3$
flow domain:	$\Omega = [0,1]\times[0,1]\times[0,1]$
circle radius:	0.15
circle center:	$(0.35, 0.35, 0.35)$
interface thickness:	$\epsilon = 1.75h$
grid resolution:	1/32, 1/64, 1/128, 1/256 and 1/512
employed methods:	LS & CLSVOF
boundary conditions:	no-slip

to be paid for in computing time, so that the LS method takes twice as long as the CLSVOF method on the fine grid, where it produces reasonable results. Still, even on the fine grid, the mass loss of the LS method is a severe problem, so that the initial circle can be hardly recovered. This mass loss intensifies on the coarse grid, where the LS method no longer produces results comparable to the CLSVOF method.

Additionally, our comparison with results from the literature shows that, although our CLSVOF method already produces good results, there is room for improvement and the possibility to adapt the CLSVOF method for a specific purpose. With our article, we hope to lay the basis for a thorough understanding and implementation of the basic CLSVOF method so that further improvements or specialties are straightforward to integrate.

All in all, the CLSVOF method compares favorably with our well-tested LS method for which convergence results have already been previously established [6, 7, 5], while maintaining a much larger amount of mass.

8.2. Three-dimensional deformation of a sphere. Our second advection test case is the fully three-dimensional deformation of a sphere. We aim to ensure that the CLSVOF method also works well in 3D, while outperforming the LS method concerning the conservation of mass. As in the previous section, the deformation velocity field is multiplied by $\cos(\pi t/T)$ which causes the flow field to return to its initial state at $t = T$. In this configuration, we have a maximum deformation and thinning at $t = 1.5$ which is well suited for an evaluation of our methods. Additionally, when the flow field reverses we can judge the LS and the CLSVOF method by the accuracy with which they recover the initial configuration. In this test, we also measure the amount of mass maintained at $t = T$ quantitatively with the VisIt software and compute convergence rates for both methods.

8.2.1. Setup of the numerical experiment. The flow domain Ω is a square with edge length 1. Moreover, a sphere of radius 0.15 is centered at $\bar{\boldsymbol{x}}(t = 0) = (0.35, 0.35, 0.35)^\mathsf{T}$ which separates both fluid phases. In the following, we denote the area occupied by the sphere as $\Omega_1 = \Omega_1(t) \subset \Omega$. The 3D deformation field is defined by

$$\begin{aligned} u &= 2\sin^2(\pi x)\sin(2\pi y)\sin(2\pi z)\cos(\pi t/T) \\ v &= -\sin^2(\pi y)\sin(2\pi x)\sin(2\pi z)\cos(\pi t/T) \\ w &= -\sin^2(\pi z)\sin(2\pi x)\sin(2\pi y)\cos(\pi t/T) \end{aligned} \tag{8.3}$$

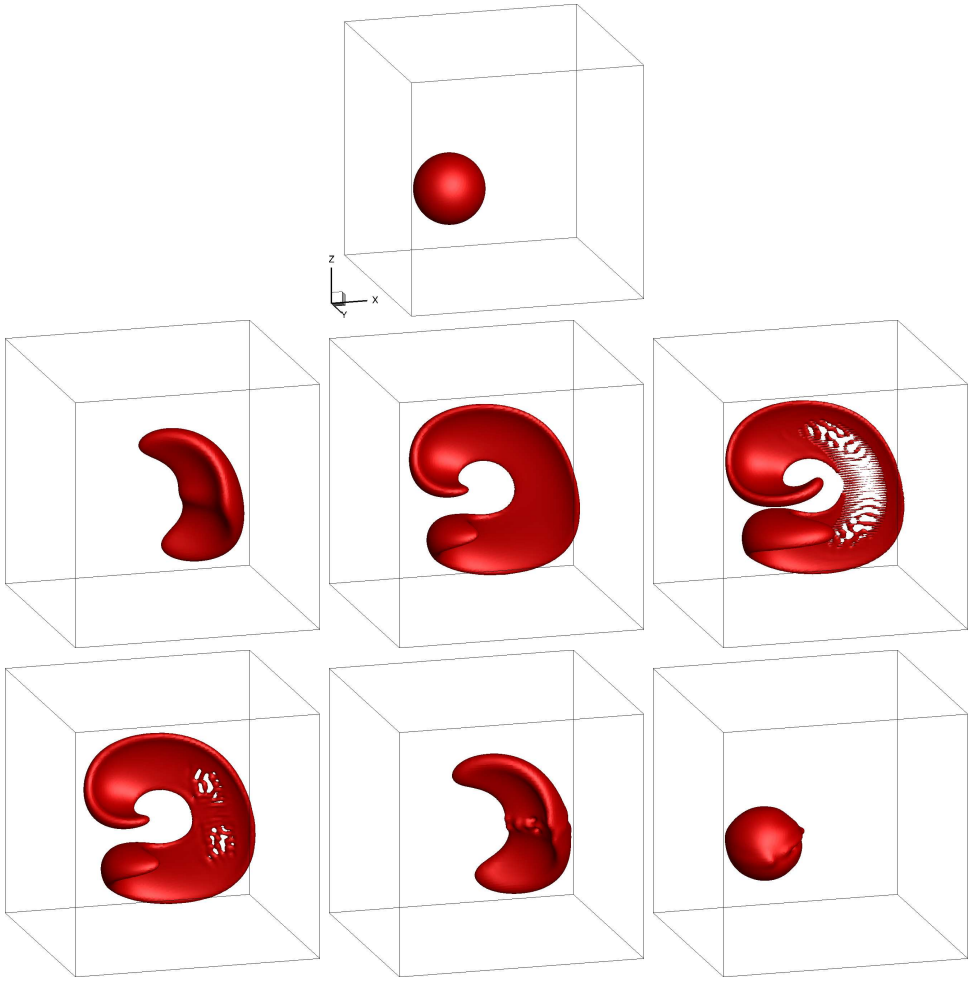


Fig. 8.3: Evolution of the three-dimensional sphere computed with the CLSVOF method on mesh size 1/128 at times 0.0, 0.48, 0.98, 1.49, 1.98, 2.48 and 3.0.

with $T = 3$ the time at which the sphere should have returned to its initial position. In Figure 8.3, the movement and deformation of the sphere over time is shown. Here, we see that the sphere is deformed by two rotating vortices which initially stretch out opposite sides of the sphere and then reverse them back to their initial shape. Note that our test case originates from a two-dimensional setting first proposed by LeVeque [18].

All relevant simulation parameters are summarized in Table 8.3, and on all grids the time step is restricted by (8.2) with a CFL number $\text{cfl} = 0.25$. Again for this test, the velocity field is prescribed, so we do not solve the full Navier-Stokes equations. As in the 2D variant above, we are only concerned with the solution of the transport equation and the reinitialization of the LS function. Our numerical experiment is evaluated as follows. First, we compare the zero level-set for both the LS and the CLSVOF method on grids with mesh sizes 1/128 and 1/256. Then, we investigate the mass convergence behavior of the LS and the CLSVOF method, i.e. we investigate

how much mass can be conserved with both methods at the final time $T = 3$. To this end, we employ five grids with mesh sizes 1/32, 1/64, 1/128, 1/256 and 1/512 which correspond to the levels $l = 1, \ldots, 5$, respectively. The analytical mass of the sphere with the parameters in Table 8.3 is $m_a = 4/3\pi r^3 \approx 0.0141374$. Numerically, we approximate the mass retained at $t = 3$ in a post-processing step with the VisIt software [2]. From the difference between the numerical and the analytical solution, we compute the discrete error norm e and the convergence rate ρ by

$$e^l = |m_l - m_a| \text{ and } \rho^l = \frac{\ln\left(\frac{e^{l-1}}{e^l}\right)}{\ln\left(\frac{h^{l-1}}{h^l}\right)} = \frac{\ln\left(\frac{e^{l-1}}{e^l}\right)}{\ln(2)} \tag{8.4}$$

since $h_l = 2h_{l+1}$ for the discrete mesh width.

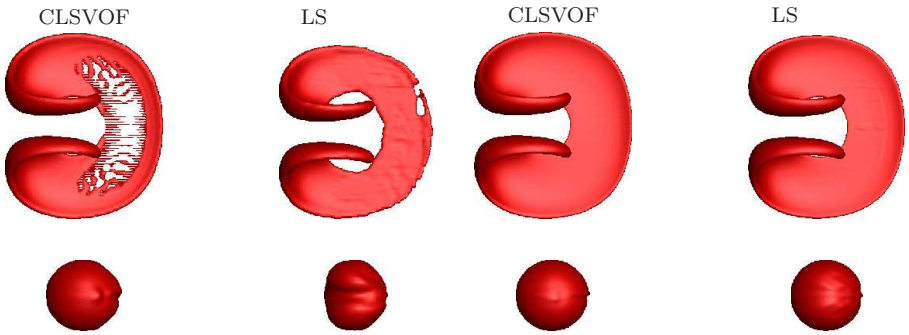
8.2.2. Expected Results. Certainly, concerning the conservation of mass, we expect that the LS method will perform less well than the CLSVOF method on all grids. However, small mass losses may still occur with the CLSVOF method due to the truncation of the volume fraction to satisfy $0 \leq F \leq 1$ at all times. On a fine enough grid, both the LS method and the CLSVOF method should be able to conserve close to 100% of the mass.

In addition, both methods have to show a substantial improvement in their ability to conserve the initial mass of the sphere on successively refined grids. Note that the operator splitting method for the transport of both the LS and the CLSVOF method is second order accurate in space and time. Additionally, the reinitialization of the LS method benefits from a fifth order WENO scheme in space and a third-order Runge-Kutta scheme in time and no interface reconstruction is necessary. Therefore, we expect a quadratic convergence rate with the LS method. In contrast, we perform a piecewise linear interface reconstruction (PLIC) in every time step of the CLSVOF method, which might result in a lower convergence rate.[5]

8.2.3. Description of Results. In a first step, we plot the deformed sphere at times $t = 1.5$ and $t = 3.0$ on two grids with mesh sizes 1/128 and 1/256, which is shown in Figure 8.4 for both the CLSVOF and the LS method. The deformed shapes at $t = 1.5$ correspond to maximum stretching, while at $t = 3.0$ the shape has returned to its initial position.

On the coarser grid (Fig. 8.4(a)), the CLSVOF method shows a certain improvement in mass and shape conservation compared to the LS method. At $t = 1.5$, the overall shape of the deformed sphere is very well maintained with slight mass loss in the inner part. Here, the CLSVOF method only partially resolves the thin interface produced at the middle section of the stretched shape. In contrast, the main mass loss with the LS function occurs at the boundary of the shape which becomes completely fragmented and even unsymmetrical. All in all, for both methods, the grid resolution is not sufficient to resolve the thin membrane stretched at the center part of the deformed shape. Consequently, in the final time frame, we see deviations from the initial spherical shape. Due to the larger mass loss, these deviations are more

[5]Note that our CLSVOF method is an essential yet still basic tool. Several enhancements such as unsplit advection [19], increased accuracy by higher-order interface reconstruction [29] or even a completion by the CLSMOF method [14] are possible. All of these, however, presuppose a thorough understanding and implementation of the basics of the CLSVOF method in the first place which we provide in this article.



(a) $h = 1/128$ **(b)** $h = 1/256$

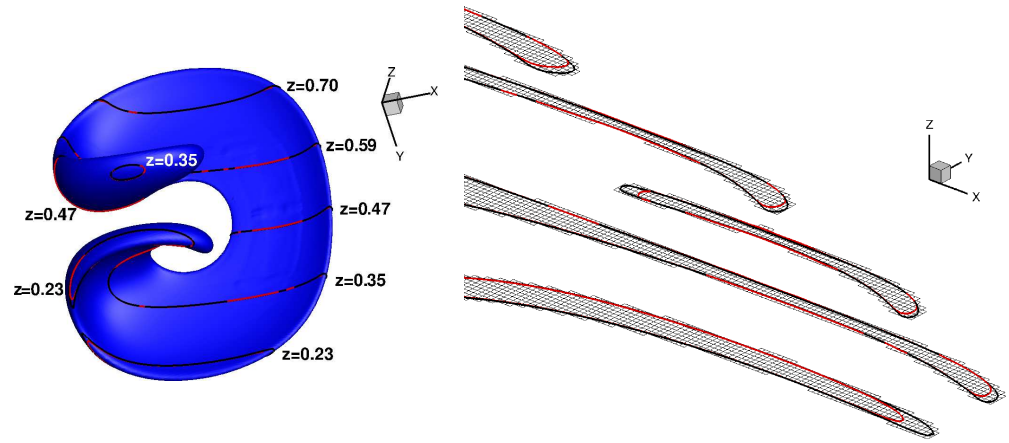
Fig. 8.4: Three-dimensional sphere deformation with the CLSVOF (left) and the LS method (right) for two grids with mesh widths 1/128 and 1/256. For both resolutions, $t = 1.5$ in the top and $t = 3.0$ in the bottom row. Ideally at time $t = 3.0$, the deformed sphere should have returned to its original shape and location; see Fig. 8.5 for a comparison of the contour lines for mesh width 1/256.

severe for the LS than for the CLSVOF method. For both, a spherical object with a 'scar' in the middle develops.

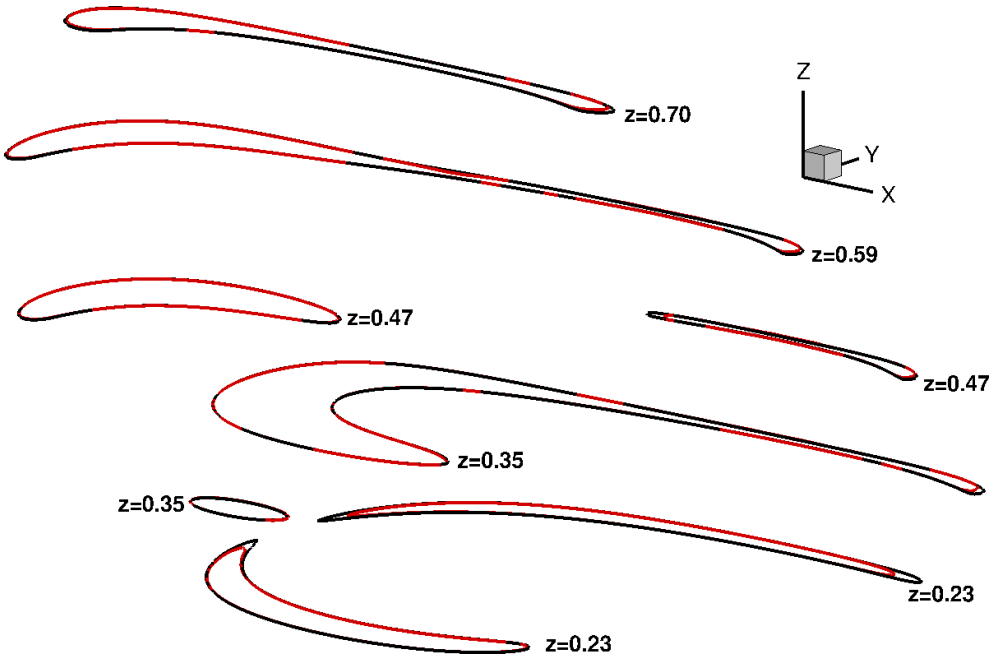
If we turn to the higher resolution (Fig. 8.4(b)), we see that both the LS and the CLSVOF method behave similarly. In order to see the differences between the CLSVOF and the LS method, we draw the zero level-set on various slices along the z-axis in Figure 8.5(b) at $t = 1.5$. Here we observe that the LS method still loses a little bit of mass at the edges, while the CLSVOF method is now able to resolve the thin structure which only consists of 2–4 grid cells in width (Fig. 8.5(a)). Then, at $t = 3.0$ in Figure 8.4(b), the scar in the final shape is again visible with the CLSVOF method. This is due to the anticipated accumulated errors of the interface reconstruction process. Similar but more severe deformations occur for the LS method due to its mass loss. However, both methods are able to return the sphere to its initial shape and location.

Second, we quantify the mass loss of both methods. In Table 8.4, the percentage of the still remaining mass at $t = 3.0$ is given. Most notably, this is a very hard test case for the LS method: On the coarsest grid, no mass is left at all. Furthermore, we lose nearly all of the mass on level 2 and about 25% of mass on level 3. Even on the finest grid, the mass cannot be preserved up to 100%. In contrast, the CLSVOF method loses only 15% of mass on the coarsest grid, and from level 3 upwards, the CLSVOF method conserves up to 100% of mass. By comparison, we see that the CLSVOF method on level 1 outperforms the LS method on level 3 by 10% concerning mass conservation. Additionally, on the finer grids, the CLSVOF method conserves almost all of the mass, which the LS method is unable to achieve.

Let us consider the effects of the LS and the CLSVOF method on the overall mass convergence behavior in space and time at $t = 3.0$ by comparing the errors and convergence rates with (8.4). Our results are summarized in Table 8.4 and Figure 8.6 where we see second order convergence for the LS method and a convergence order of about 1.4 for the CLSVOF method. As expected, the convergence rate is slightly



(a) Zero isosurface of the deformed sphere with indication of the extracted slices (left) and the number of grid cells resolving the filaments on the slices (right).



(b) Zero contour of both the LS (red) and the CLSVOF method (black) on the slices normal to the z-axis indicated above.

Fig. 8.5: Three-dimensional sphere deformation with the CLSVOF (black) and the LS method (red) for the grid with mesh width 1/256 (cf. Fig. 8.4(b)). Shown are the zero level-set contour of both the LS and the CLSVOF method on various slices normal to the z-axis at $t = 1.5$.

Table 8.4: Table of mass convergence for the three dimensional deformation of the sphere with the LS and the CLSVOF method evaluated at $t = 3.0$; see Fig. 8.6 for a convergence plot.

	LS			CLSVOF		
Level	Mass [%]	e^l_{LS}	ρ^l_{LS}	Mass [%]	e^l_{CLSVOF}	ρ^l_{CLSVOF}
1	0	1.414_{-2}	–	85.8	2.007_{-3}	–
2	13.2	1.227_{-2}	0.204	95.2	6.777_{-4}	1.566
3	74.5	3.605_{-3}	1.767	99.3	1.029_{-4}	2.720
4	94.0	8.427_{-4}	2.097	99.7	3.657_{-5}	1.492
5	98.5	2.078_{-4}	2.020	99.9	1.377_{-5}	1.409

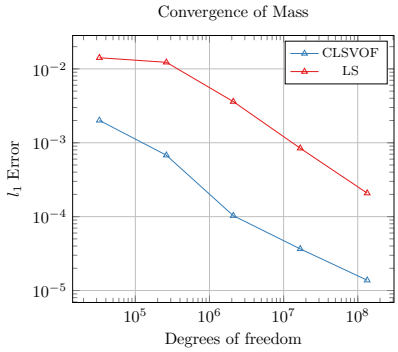


Fig. 8.6: Mass convergence history with the LS and the CLSVOF method for the three-dimensional deformation of a sphere; compare Table 8.4.

worse for the CLSVOF method. However, the absolute error in mass on the coarsest grid for the CLSVOF method is smaller than the error of the LS method on level 3. Furthermore, on the finest grid, the absolute error is still about an order of magnitude smaller with the CLSVOF than with the LS method.

8.2.4. Comparison to results from the literature. Last, we compare our results to those from the literature. In Figure 8.7, for a resolution $h = 1/128$, we compare our results (red) with those by Wang et al. (violet) [39] and with those by Ménard, Tanguy and Berlemont (blue) [23] at $t = 1.5$ and $t = 3.0$ with a resolution $h = 1/150$. All three results are very close to each other and mainly differ in their ability to resolve the thin membrane in the middle of the droplet. Here, the CLSVOF method by Ménard performs best since it is specifically adapted to handle very thin filaments for the simulation of the primary break-up of liquid jets. The blue results as presented in [40] are obtained by a CLSVOF method which uses a Lagrangian method with a second-order Runge-Kutta scheme for the advection of the piecewise linear interface. These results compare very well with our own work. However, the thin structure is recovered slightly better in [40].For all CLSVOF methods, a scar is present in the sphere in the final time frame, which is probably due to slight mass loss and accumulated numerical errors in the interface reconstruction process [39].

Again, from a qualitative point of view, all three results are in very good agree-

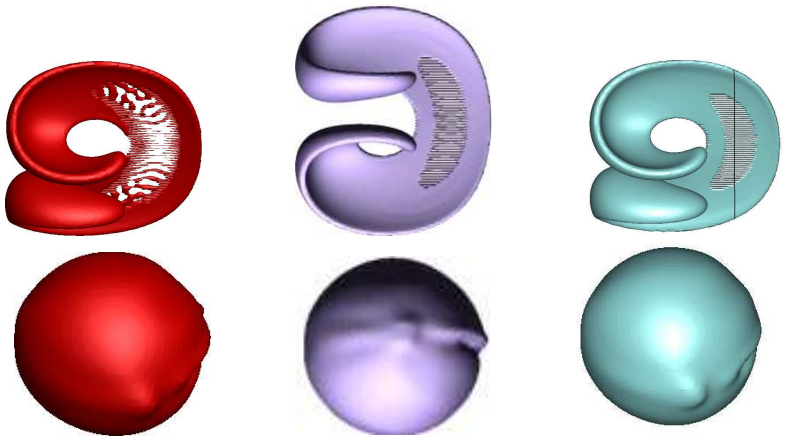


Fig. 8.7: Three-dimensional sphere deformation with our solver NaSt3DGPF (red), results from [39] (violet) and from [23] (blue) at times $t = 1.5$ (top) and $t = 3.0$ (bottom). Unfortunately, the results in [39] and [23] employ different perspectives.

ment with each other and the slight differences in mass conservation can be attributed to the applied methods of transport, reinitialization and interface reconstruction.

8.2.5. Remarks. In this subsection, we established that both the LS and the CLSVOF method fare very well with our three-dimensional test case. From level 3 onwards both methods are able to return the deformed shape back to its initial position and to its spherical state. From level 4 onwards, both methods conserve a substantial amount of mass. Furthermore, we established the expected second-order convergence rate for the LS and a convergence order 1.4 for the CLSVOF method. Despite the better rate of the LS method, the mass error with the CLSVOF method was always about 1 to 3 orders of magnitude smaller. Therefore, and looking back at Figure 8.6, it seems unlikely that the point where the LS method outperforms the CLSVOF method concerning mass conservation becomes relevant – even with today's computing power.

All in all, our results are very much in favor of the CLSVOF method. Also in three dimensions, this method is well equipped to deal with severe topological changes and to maintain the mass of the thin filaments developing on the scale of the mesh size. Furthermore, this test case strongly supports the 3D suitability of our new vertex finding algorithm in the re-distancing of the CLSVOF method.

Thus, we are perfectly equipped to move on to test cases which involve the whole flow solver; see [17] for first steps into that direction.

9. Conclusion. We implemented the CLSVOF method, which proved to be faster than our previous higher-order LS method and conserved mass excellently even on coarse grids. In our description of the CLSVOF method, we included all details of the implementation. Thus, this article can be used as a comprehensive guide for an implementation into existing LS Navier-Stokes solvers, which has not been available so far. Additionally, we presented a new technique for the reinitialization of the LS function within the CLSVOF method. Furthermore, we also addressed the details of parallelization, which is neglected in the standard literature. Here, we focused on the parallelization of our new vertex finding algorithm, which is a key part of the reinitialization whose parallelization is not straightforward. In addition to the LS method,

the CLSVOF method was validated for two basic advection tests in two and three dimensions. To summarize, the CLSVOF method showed superb mass conservation properties and performed computations faster than our high-order LS method whose reinitialization is very costly.

Of course, the CLSVOF method can always be adapted and improved to fit to a specific simulation purpose. Ménard, Tanguy and Berlemont [23] specifically enhance the interface reconstruction process to handle very thin filaments for the simulation of the primary break-up of liquid jets. In [40], a second-order Lagrangian method is chosen for the transport of the volume fractions for the simulation of liquid-liquid drop impact and of plunging breaking waves. Furthermore, unsplit methods for the transport of the LS and the VOF function are considered in [19]. In an unsplit method, the interface has to be reconstructed only once, which is invaluable if we aim at expensive higher-order interface reconstruction schemes in three-dimensions. A further development of the CLSVOF method is the coupled level-set and moment-of-fluid (CLSMOF) method, which uses next to the LS and the VOF function also a reference centroid in order to produce a slope and an intercept for the local reconstruction of the interface [14].

Nevertheless, all these adaptions and improvements rely on a thoroughly implemented basic CLSVOF method, for which we have provided a framework in this article.

REFERENCES

[1] J. Brackbill, D. Kothe, and C. Zemach, *A continuum method for modeling surface tension*, Journal of Computational Physics, 100 (1992), pp. 335–354.

[2] H. Childs, E. Brugger, B. Whitlock, J. Meredith, S. Ahern, D. Pugmire, K. Biagas, M. Miller, C. Harrison, G. Weber, H. Krishnan, T. Fogal, A. Sanderson, C. Garth, E. Bethel, D. Camp, O. Rübel, M. Durant, J. Favre, and P. Navrátil, *VisIt: An end-user tool for visualizing and analyzing very large data*, in High Performance Visualization–Enabling Extreme-Scale Scientific Insight, Oct 2012, pp. 357–372.

[3] A. Chorin, *Numerical solutions of the Navier-Stokes equations*, Mathematics of Computations, 22 (1968), pp. 745–762.

[4] R. Croce, *Ein paralleler, dreidimensionaler Navier-Stokes-Löser für inkompressible Zweiphasenströmungen mit Oberflächenspannung, Hindernissen und dynamischen Kontaktflächen*, Diplomarbeit, Institut für Angewandte Mathematik, Universität Bonn, 2002.

[5] ———, *Numerische Simulation der Interaktion von inkompressiblen Zweiphasenströmungen mit Starrkörpern in drei Raumdimensionen*, Dissertation, Institut für Numerische Simulation, Universität Bonn, 2010.

[6] R. Croce, M. Griebel, and M. Schweitzer, *A parallel level-set approach for two-phase flow problems with surface tension in three space dimensions*. Preprint 157, Sonderforschungsbereich 611, Universität Bonn, 2004.

[7] ———, *Numerical simulation of bubble and droplet deformation by a level set approach with surface tension in three dimensions*, International Journal for Numerical Methods in Fluids, 62 (2010), pp. 963–993.

[8] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell, *A hybrid particle level set method for improved interface capturing*, Journal of Computational Physics, 183 (2002), pp. 83–116.

[9] M. Griebel, T. Dornseifer, and T. Neunhoeffer, *Numerical simulation in fluid dynamics: A practical introduction*, vol. 3, SIAM, Philadelphia, 1998.

[10] S. Gross, *Numerical methods for three-dimensional incompressible two-phase flow problems*, Dissertation, IGPM, RWTH Aachen, 2008.

[11] M. Herrmann, *Refined level set grid method for tracking interfaces*, Annual Research Briefs, Center for Turbulence Research, NASA Ames/Stanford University, (2005), pp. 3–18.

[12] C. Hirt and B. Nichols, *Volume of fluid VOF method for the dynamics of free boundaries*, Journal of Computational Physics, 39 (1981), pp. 201–225.

[13] INSGrid, *High-Performance Cluster Computers at the INS and SFB 1060*, 2013. `http://wissrech.ins.uni-bonn.de/research/atacama/`.

[14] M. Jemison, E. Loch, M. Sussman, M. Shashkov, M. Arienti, M. Ohta, and Y. Wang,

A coupled level set-moment of fluid method for incompressible two-phase flows, Journal of Scientific Computing, 54 (2013), pp. 454–491.

[15] R. Keck, *Reinitialization for level-set methods*, Diplomarbeit, Fachbereich Mathematik der Universität Kaiserslautern, 1998.

[16] M. Klitz, *Homogenised fluid flow equations in porous media with application to permeability computations in textiles*, Diplomarbeit, Institut für Numerische Simulation, Universität Bonn, 2006.

[17] M. Klitz, *Numerical Simulation of Droplets with Dynamic Contact Angles*, Dissertation, Institut für Numerische Simulation, Universität Bonn, Dec. 2014.

[18] R. LeVeque, *High-resolution conservative algorithms for advection in incompressible flow*, SIAM Journal on Numerical Analysis, 33 (1996), pp. 627–665.

[19] P. Liovic, M. Rudman, J.-L. Liow, D. Lakehal, and D. Kothe, *A 3D unsplit-advection volume tracking algorithm with planarity-preserving interface reconstruction*, Computers & Fluids, 35 (2006), pp. 1011–1032.

[20] W. Lorensen and H. Cline, *Marching cubes: A high resolution 3d surface construction algorithm*, in ACM Siggraph Computer Graphics, vol. 21, ACM, 1987, pp. 163–169.

[21] F. Losasso, R. Fedkiw, and S. Osher, *Spatially adaptive techniques for level set methods and incompressible flow*, Computers & Fluids, 35 (2006), pp. 995–1010.

[22] T. Ménard, *Développement d'une méthode Level Set pour le suivi d'interface – Application à la rupture de jet liquide*, PhD thesis, Université de Rouen, 2007.

[23] T. Ménard, S. Tanguy, and A. Berlemont, *Coupling level set/VOF/ghost fluid methods: Validation and application to 3D simulation of the primary break-up of a liquid jet*, International Journal of Multiphase Flow, 33 (2007), pp. 510–524.

[24] NaSt3DGPF, *A parallel 3d free surface flow solver*. `http://wissrech.iam.uni-bonn.de/research/projects/NaSt3DGPF/`.

[25] S. Osher and J. A. Sethian, *Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations*, Journal of Computational Physics, 79 (1988), pp. 12–49.

[26] ParaView, *ParaView User's Guide (v3.10)*, 2011. `http://www.paraview.org/`.

[27] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang, *A PDE-based fast local level set method*, Journal of Computational Physics, 155 (1999), pp. 410–438.

[28] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical recipes 3rd edition: The art of scientific computing*, Cambridge University Press, 2007.

[29] Y. Renardy and M. Renardy, *PROST: A parabolic reconstruction of surface tension for the volume-of-fluid method*, Journal of Computational Physics, 183 (2002), pp. 400–421.

[30] W. Rider and D. Kothe, *Stretching and tearing interface tracking methods*, AIAA paper, 95 (1995), pp. 806–816.

[31] G. Russo and P. Smereka, *A remark on computing distance functions*, Journal of Computational Physics, 163 (2000), pp. 51–67.

[32] G. Son, *Efficient implementation of a coupled level-let and volume-of-fluid method for three-dimensional incompressible two-phase flows*, Numerical Heat Transfer, Part B: Fundamentals, 43 (2003), pp. 549–565.

[33] G. Son and N. Hur, *A coupled level set and volume-of-fluid method for the buoyancy-driven motion of fluid particles*, Numerical Heat Transfer, Part B: Fundamentals, 42 (2002), pp. 523–542.

[34] M. Sussman, *A second order coupled level set and volume-of-fluid method for computing growth and collapse of vapor bubbles*, Journal of Computational Physics, 187 (2003), pp. 110–136.

[35] M. Sussman and E. Fatemi, *An efficient, interface preserving level set re-distancing algorithm and its application to interfacial incompressible fluid flow*, SIAM Journal on Scientific Computing, 20 (1999), pp. 1165–1191.

[36] M. Sussman and E. Puckett, *A coupled level set and volume-of-fluid method for computing 3D and axisymmetric incompressible two-phase flows*, Journal of Computational Physics, 162 (2000), pp. 301–337.

[37] M. Sussman, K. Smith, M. Hussaini, M. Ohta, and R. Zhi-Wei, *A sharp interface method for incompressible two-phase flows*, Journal of Computational Physics, 221 (2007), pp. 469–505.

[38] Tecplot, *Tecplot 360 users manual*, 2013. `http://www.tecplot.com/`.

[39] Z. Wang, J. Yang, B. Koo, and F. Stern, *A coupled level set and volume-of-fluid method for sharp interface simulation of plunging breaking waves*, International Journal of Multiphase Flow, 35 (2009), pp. 227–246.

[40] Z. Wang, J. Yang, and F. Stern, *Comparison of particle level set and CLSVOF methods for interfacial flows*, in 46th AIAA Aerospace Sciences Meeting and Exhibit, 2008, pp. 7–10.